October 1977

This document describes the purpose and function
of the FORTRAN/RT-11 language extensions for
real-time support, and FDT, the FORTRAN de-
bugging technique.

FORTRAN/RT-11
Extensions Manual

Order No. AA-2124D-TC

digital equipment corporation · maynard. massachusetts

The postage prepaid READER'S COMMENTS form on the last page of this
document requests the user's critical evaluation to assist us in pre-
paring future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECSYSTEM-20	TYPESET-11

CONTENTS

CONTENTS (CONT.)

CONTENTS (CONT.)

FIGURES

TABLES

v

PREFACE

The <u>RT-11 FORTRAN Extensions Manual</u> describes FORTRAN-callable real-time support routines, and FDT, the FORTRAN debugging technique.

RT-11 is documented in the following manuals:

<u>Introduction to RT-11</u> (DEC-11-ORITA-A-D)
<u>RT-11 Pocket Guide</u> (DEC-11-ORRCB-A-D)
<u>RT-11 System User's Guide</u> (DEC-11-ORGDA-A-D)
<u>RT-11 Advanced Programmer's Guide</u> (DEC-11-ORAPA-A-D)
<u>RT-11 Documentation Directory</u> (DEC-11-ORDDB-A-D)
<u>RT-11 System Message Manual</u> (DEC-11-ORMEB-A-D)
<u>RT-11 System Generation Manual</u> (DEC-11-ORGMB-A-D)
<u>RT-11 System Release Notes</u> (DEC-11-ORNRB-A-D)

FORTRAN is documented in the following manuals:

<u>RT-11/RSTS/E FORTRAN IV User's Guide</u> (DEC-11-LRRUB-A-D)
<u>PDP-11 FORTRAN Language Reference Manual</u> (DEC-11-LFLRA-C-D and
    DEC-11-LFLRA-C-DN1)
<u>RT-11 FORTRAN IV Installation Guide</u> (DEC-11-LRSIA-A-D)

The three sets of laboratory peripherals supported by the FORTRAN extensions are described in the following sets of manuals. You should be familiar with the function of the peripherals hardware on your system, and with the terms used in these manuals.

1. <u>AD11-K Analog to Digital Converter User Manual</u>
    <u>(EK-AD11K-OP-001)</u>
    <u>AM11-K Multiple Gain Multiplexer User's Manual</u>
    <u>(EK-AM11K-TM-002)</u>
    <u>AA11-K 4-channel D/A and Display Control User's Manual</u>
    <u>(EK-AA11K-TM-001)</u>
    <u>DR11-K Interface User's Guide and Maintenance Manual</u>
    <u>(EK-DR11K-MM-001)</u>
    <u>KW11-K Dual Programmable Real Time Clock User Manual</u>
    <u>(EK-KW11K-OP-001)</u>

2. <u>AR11 User's Guide</u> (DEC-11-HARUG-B-D)

3. <u>LPS-11 Laboratory Peripheral System User's Guide</u>
    <u>(EK-LPS11-OP-003)</u>
    <u>LPS11-S Laboratory Peripheral System Maintenance Manual</u>
    <u>(EK-LPS11-MM-002)</u>
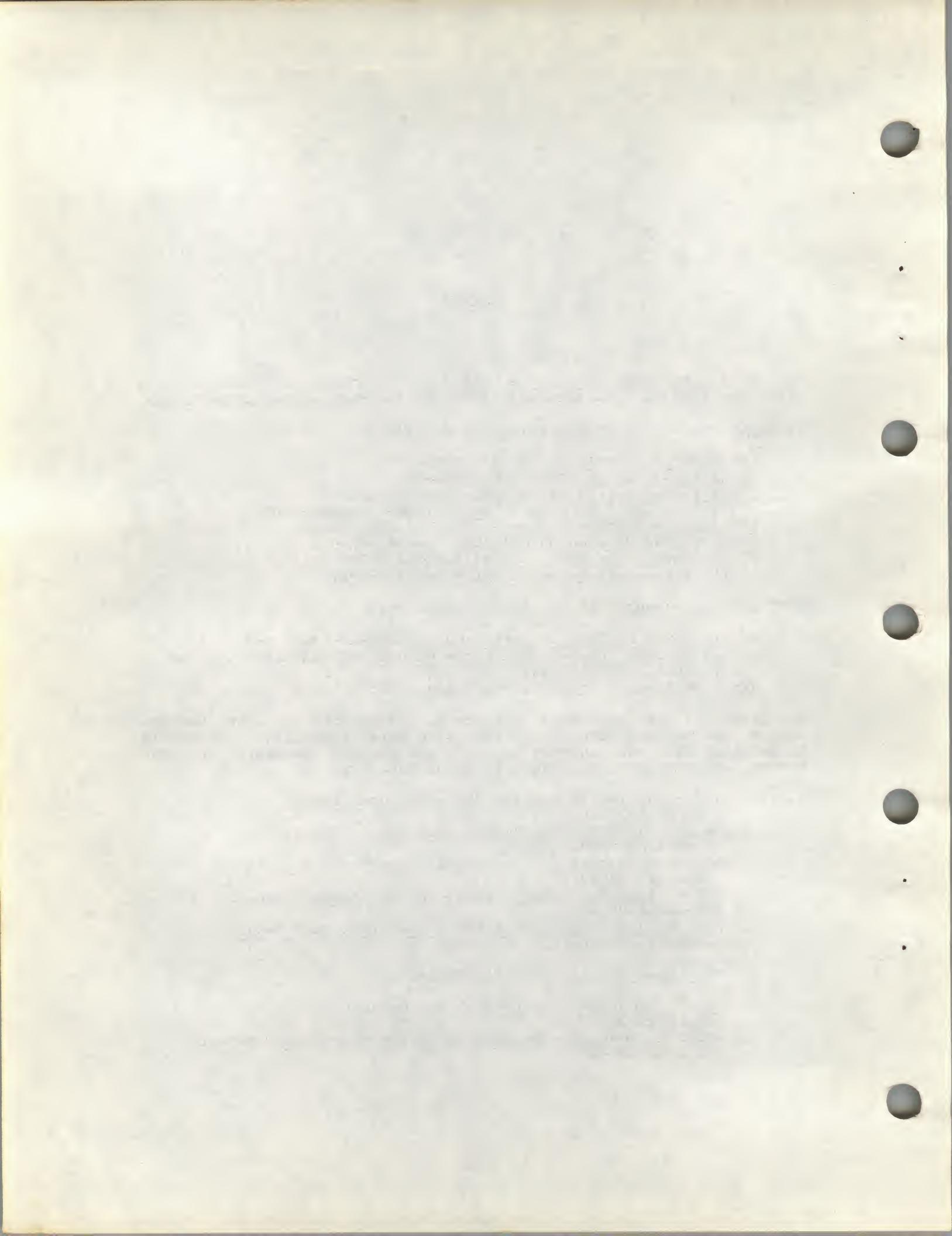
CHAPTER 1

RT-11 FORTRAN EXTENSIONS

1.1 WHAT ARE THE FORTRAN EXTENSIONS?

The RT-11 FORTRAN Extensions are a library of routines for real-time
data acquisition and processing. The extensions themselves are
subprograms written in MACRO assembler language and callable from
FORTRAN. The extensions allow you to control a programmable clock
counter and laboratory peripherals from a FORTRAN program.

Specifically, the FORTRAN real-time extensions support any of the
following sets of peripherals hardware:

  ● the K-series hardware (AD11-K analog-to-digital converter,
    AM11-K multiple gain multiplexer, AA11-K digital-to-analog
    converter, DR11-K digital interface, and KW11-K programmable
    clock counter)

  ● the AR11 real-time analog system

  ● the LPS11 laboratory peripheral system

See Section 1.5 for a discussion of system configuration.

The FORTRAN extensions can be combined to provide five kinds of
operations: time interval control and sampling, digital input and
output, analog-to-digital conversion and sampling, digital-to-analog
conversion, and graph plotting. The following section describes the
functions available and lists the routines you use to perform those
functions. Detailed instructions for using each routine appear in
Sections 1.4.1 through 1.4.20.

Time-Interval Control and Sampling

    Timed program delay.
        Use SETR in single-interval mode to specify the length of
        the delay interval, and LWAIT to detect the end of the
        interval.

    Schedule an event at the completion of a time interval.
        Use SETR to request a timed program delay, and to designate
        a completion routine to execute at the end of the delay.

    Repeated time intervals.
        Use SETR in repeated-interval mode to specify the duration
        of the intervals. The clock counter generates an interrupt
        at the end of each interval.

Schedule periodic events.
> Use SETR to request repeated time intervals, and to designate a completion routine that executes at the completion of each interval.

Time stamping.
> Use IDIR or IDOR to read the current value of the software clock (see Section 1.2.2, which describes the software clock). You can also use DRS to read the software clock during digital sampling. Use FLT16 to convert the unsigned binary integer time value to a real number.

Measuring the time intervals between events.
> Use HIST to read the time and SETR to run the clock in external-event timing mode. Whenever an external event occurs, HIST reads the time intervening since the last external event. (You cannot do this with an AR11 because its clock does not have the external event timing mode.)

Measuring the latencies of a series of events relative to an initial event.
> Use HIST to read the clock value at the time of the event, and SETR to run the clock in external-event timing from zero mode. (You cannot do this with an AR11 because the AR11 clock does not have the required timing mode.)

Digital Input and Output

Read digital input.
> Use IDIR to read the value of the digital input register. IDIR accepts data in either binary or BCD format and stores it in binary format.

Generate digital output.
> Use IDOR to load a digital output register. Use the conversion routine KB2BCD to generate BCD format data for output.

Digital sampling at regular time intervals.
> Use DRS to sample from a digital input register, and SETR in repeated-interval mode to generate clock interrupts at regular intervals (thus triggering the DRS sampling).

Continuous digital sampling at regular time intervals.
> Request digital sampling at regular time intervals using DRS and SETR, and designate a completion routine to process the data in the call to SETR.

Digital sampling on external events.
> Use DRS to sample from a digital input register.

Continuous digital sampling on external events.
> Request digital sampling on external events using DRS, and designate a completion routine to process the data in the call to DRS.

Analog-to-Digital (A/D) Conversion and Sampling

Read one value from an analog input.
> Use IADC to take a single sample from an A/D channel, and CVSWG to convert the digitized value from a nonstandard integer data format to a real number (if autogain or switched gain was used).

Read a series of values from one or more analog input channels.
Use SETR to set the sampling rate with the clock in repeated-interval mode, and RTS to collect a single sweep of A/D samples. Use LWAIT to detect the completion of sampling, and CVSWG to convert the digitized values from nonstandard integer data format to real numbers.

Read values continuously from one or more analog input channels and store the values on a mass storage device.
Use SETR to set the sampling rate with the clock in repeated-interval mode, and RTS to sample A/D data continuously. Designate a completion routine to handle the data throughput to disk in the call to RTS, and use CVSWG to convert the digitized values from nonstandard integer data format to real numbers.

Digital-to-Analog (D/A) Conversion

Generate an analog voltage.
Use IDOR to generate a digital output value for a D/A converter.

Generate a series of analog voltages.
Use SETR in repeated-interval mode to control the time base, and to designate a completion routine. In the completion routine, you use IDOR to generate the digital output value for a D/A converter.

Graph Plotting

Data scaling.
Use CLRD to multiply the data in an array so that the data values are in the required range.

Single display of Y-axis data.
Use FSH to display a data array once.

Continuous display of Y-axis data (refresh scopes).

(a) Use SETR in repeated-interval mode to control the refresh interval, and to designate a completion routine. In the completion routine, you call FSH to display a data array.

(b) Use DIS to display a data array constantly, and SETR to designate a completion routine. In the completion routine, you use SDIS to stop the display.

Continuous display of Y-axis data (storage scopes).
Use FSH to display a data array once, and IDOR to control the intensify bit and erase bit of the scope controller.

Continuous display of X-Y data pairs (refresh scopes).

(a) Use SETR in repeated-interval mode to control the refresh interval and to designate a completion routine. In the completion routine, you call FXY to display the X and Y data arrays.

(b) Use DXY to display the X and Y data arrays constantly, and SETR to designate a completion routine. In the completion routine, you use SDIS to stop the display.

Continuous display of X-Y data pairs (storage scopes).
Use FXY to display the X and Y data arrays once, and IDOR to control the intensify bit and erase bit of the scope controller.

1.2 THINGS TO KNOW ABOUT RT-11 AND THE FORTRAN EXTENSIONS

There are three variants of the RT-11 monitor: RT-11 Single Job (SJ), RT-11 Foreground/Background (FB), and RT-11 Extended Memory (XM). Consult the RT-11 documentation for descriptions of the features of the monitor variants. The FORTRAN extensions run under both RT-11 SJ and RT-11 FB.

CAUTION

Do not use the FORTRAN extensions under RT-11 XM.

The FORTRAN extensions run with FORTRAN IV version 2 which includes virtual-array support. (See the FORTRAN Language Reference Manual for description of the VIRTUAL array declarator.) However, the FORTRAN extensions impose certain restrictions on the use of FORTRAN virtual arrays. The extensions routines cannot refer to virtual arrays. That is, the array arguments passed to the extensions routines cannot be virtual-array names. If your FORTRAN program passes a virtual array to an extensions routine, the results are invalid and unpredictable. There is no error message. Your FORTRAN main program can contain virtual arrays and you can copy the conventional arrays used by the extensions routines into virtual arrays.

NOTE

Virtual arrays cannot appear in COMMON or EQUIVALENCE statements.

1.2.1 Completion Routines

In the FORTRAN extensions package, completion routines are subroutines called from within an interrupt service routine. The FORTRAN extensions library provides standard interrupt service routines to handle all interrupts from the clock and laboratory peripherals. Essentially, you write your own completion routines to extend the power of interrupt service routines.

Several of the FORTRAN extensions routines allow you to designate a completion routine name, and the conditions under which you want it to execute. The idea behind completion routines is that you will want them to execute upon completion of some operation that generates interrupts--for example, at the end of data sampling. The condition that you specify is always in terms of some number of interrupts. For example, you can specify that a completion routine is to execute after a certain number of clock interrupts. Depending on the mode of clock operation, the routine executes after a defined interval, after a defined number of data samples, or after a defined number of external event interrupts.

The RT-11 monitor puts the conditions and the completion routine name into a subroutine call in the interrupt service routine. Then whenever the appropriate interrupt has occurred and the conditions are satisfied, the interrupt service routine calls the completion routine.

If no other completion routine is executing, the state of the job being executed determines what happens when a completion routine is called. Under the RT-11 SJ monitor, the completion routine executes immediately. Under the RT-11 FB monitor, foreground completion routine requests have priority over background jobs. If a foreground job requests a completion routine while a background job is executing, RT-11 FB suspends the background job, and executes the foreground completion routine. At the end of the completion routine, the monitor returns control to the interrupted background job (unless the foreground job has requested service in the meantime). If a background job requests a completion routine while a foreground job is executing, RT-11 FB queues the completion routine request until the foreground job gives up control of the system, and then executes the background completion routine.

If another completion routine is already executing, the action taken depends on the RT-11 monitor. Under the RT-11 SJ monitor, the second completion routine interrupts the first, executes, and returns to the first completion routine. This procedure works only for two different completion routines because completion routines are not reentrant. That is, a FORTRAN completion routine must not attempt to interrupt itself as the results are undefined (and there is no error message). Under RT-11 FB, completion routines do not interrupt each other. The completion routine executing continues to completion regardless of whether it was requested from the foreground or the background. Then the second completion routine executes.

Considerations in Using Completion Routines

Completion routines do not "lock out" interrupts or interrupt service routines. You do not lose interrupts or data while completion routines are running.

Completion routines increase the system overhead time in interrupt handling, and therefore decrease the rate at which interrupts can be serviced. The maximum data-acquisition speed is lower in programs that use completion routines.

Completion routines should not request input or output from a hard copy terminal. Terminal I/O results in delays in returning from completion routines.

If you anticipate that several completion routines could be active simultaneously, use the SYSLIB IQSET call to extend the RT-11 queue length.

You must declare a completion-routine name in an EXTERNAL statement. If you fail to do so, your program does not link properly.

A completion routine cannot call a subprogram that is called elsewhere in the FORTRAN program. Because completion-routine execution is asynchronous, the monitor could interrupt the subprogram to begin executing the completion routine. The completion routine would then attempt to call the interrupted subprogram, with undefined results, because the FORTRAN subprogram code is not reentrant.

1.2.2 Software Clock

The FORTRAN extensions library provides a time service known as "the software clock." The software clock is a memory location that SETR increments by 1 on each hardware clock counter interrupt. The software clock operates when, and only when, SETR is running the hardware clock in repeated-interval mode.

The rate of the software clock is controlled by the rate of clock counter overflow interrupts specified in the call to SETR. The rate of overflow interrupts is a function of the clock counter frequency (irate) and the initial count (rcount), where the interrupt rate is equal to irate/rcount. (The time interval between interrupts is equal to irate*rcount.)

The software-clock values are unsigned binary integers. Before you can do FORTRAN arithmetic with software-clock values, you must convert the values to real number form using FLT16. The software clock counts up to 65535 (octal 177777), which is the largest possible unsigned binary integer. After the next clock-overflow interrupt, the value of the software clock is zero.

The IDIR, IDOR, and DRS routines read the software clock. The IDOR routine can change the software-clock value.

1.2.3 SYSLIB Library

SYSLIB is the RT-11 system library. The services provided by SYSLIB are described in the RT-11 Advanced Programmer's Guide. For example, SYSLIB provides character string manipulation routines, double word integers, programmed requests, and a FORTRAN interrupt service facility (INTSET).

The real-time extensions software protects the appropriate device interrupt vectors during execution using the RT-11 .PROTECT programmed request. Multiple .PROTECT requests on the same vector cause a fatal system error. Therefore, you should not use the INTSET call or the .PROTECT programmed request in any program that uses the FORTRAN extensions. There is one exception: you can use INTSET or .PROTECT for the DR11-K or LPSDR-A vectors if the program does not call IDIR, IDOR, or DRS.

1.2.4 The RT-11 User Service Routine

The User Service Routine (USR) portion of the RT-11 monitor is not permanently resident in memory. The USR is normally swapped into memory when any of its services is required. (The USR supports the RT-11 file structure, and, for example, handles opening files.)

Swapping the USR can slow program execution. Programs can run faster if you lock the USR into memory before the program starts executing. The following instruction locks the USR into memory:

    SET USR NOSWAP

Locking the USR into memory is recommended because that eliminates the possibility of the USR swapping over the extensions or the completion routines. If space limitations prevent you locking the USR into memory, consult the FORTRAN documentation to find out the address where the USR swaps and consult your program link map to see if there are any conflicts.

1.2.5 Action of CTRL/C

The action of CTRL/C on a running FORTRAN program depends on which RT-11 monitor is in use.

Under RT-11 SJ, the CTRL/C causes a UNIBUS INIT, which clears and initializes all the devices.

Under RT-11 FB, the CTRL/C causes the real-time extensions software to clear all of the device registers in the device list maintained by the extensions. If your program is using support libraries other than the real-time extensions library (for example, VTLIB or LVLIB), use the monitor INIT command.

1.2.6 Programming Requirements for High-Volume Data Acquisition

The FORTRAN extensions data-acquisition routines DRS, HIST, and RTS place data in FORTRAN arrays. If the number of data points to be collected exceeds the length of the array, these routines treat the array as a circular array. That is, when the routine has filled the array, it continues to put data in the array, starting again from the beginning (see Figure 1-1). Unless your program removes data from the array, the routine eventually attempts to overwrite data. When it attempts to overwrite data, the routine sets a flag for this error condition, which is called "data overrun."

Linear FORTRAN Array
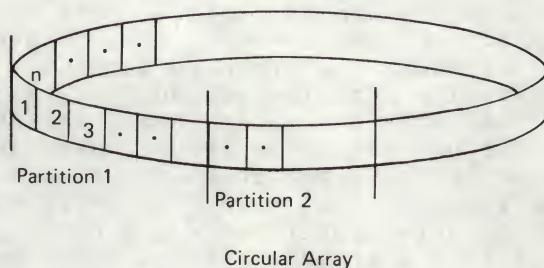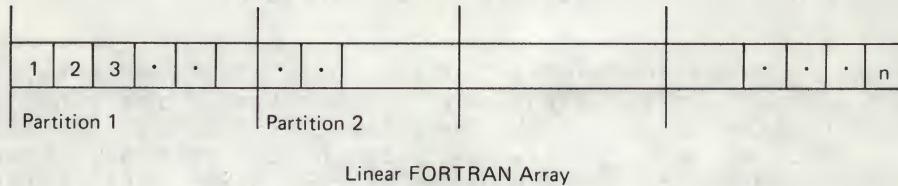
Circular Array

Figure 1-1 Linear and Circular Arrays
The linear FORTRAN array wraps around to form a
circular array. DRS and RTS treat groups of array
elements as array partitions.

There are four related variables involved in a circular array structure: the pointer for adding the next data point, the space remaining before data overrun, the pointer to the next data point to remove, and the space containing valid data (see Figure 1-2). In the FORTRAN extensions routines, space can be expressed in units of array elements or in units of array partitions, where a partition is a subset of the total array length.
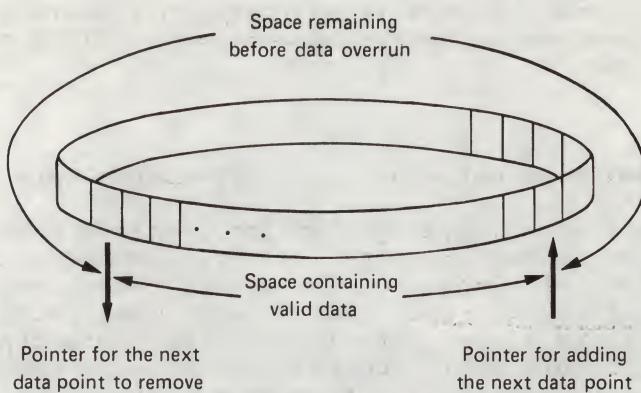
Figure 1-2 Circular Array Pointers
A circular array structure involves two array pointers and available space variables.

The extensions routines handle adding data to the array but not removing data; your FORTRAN program must handle removing data. The extension routine maintains the pointer for adding the next data point; this pointer is inaccessible to your program. Your program must define and maintain the pointer to the next data point to be removed.

Both your program and the extension routine maintain the variable containing the space remaining before data overrun (the available space variable). HIST defines the available space variable as the number of array elements before data overrun. HIST decrements the available space variable each time it adds a data point to the array. DRS and RTS define the available space variable as the number of complete array partitions available before data overrun. DRS and RTS decrement the available space variable each time an array partition is filled. Each time your program removes an array element (with HIST) or the contents of an array partition (with DRS or RTS) thereby making space available, it must increment the available space variable.

Your program is also responsible for ensuring that the data-removal pointer does not pass the data-addition pointer. Your program ensures proper data removal by starting to remove data only after the extension routine informs your program that data have been received. (In practice, you can either monitor the available space pointer with LWAIT and remove data when it changes in value, or arrange to have a completion routine execute after a defined number of interrupts.)

1.3 CONVENTIONS

The following sections define the syntax conventions and notation conventions used in this chapter.

1.3.1 Syntax Conventions

The syntax conventions described below apply to all of the subroutine and function calls appearing in the chapter. The discussion of syntax conventions refers to the following sample sequence:

```
CALL SUBR([nvalues#],rvalue[,[iflag*]][,[rvalue2],call])
```

Any capitalized words or letters are obligatory. You must specify them in exactly the same way in all calls. For example, CALL SUBR would be the required portion of the sequence above.

Punctuation is obligatory except when enclosed in square brackets.

Any words or letters in lower case indicate variable parts of the syntax. You can replace these token items with names of your choosing that satisfy other constraints of the syntax. As far as practical, the tokens are mnemonic for the function of the argument required. For example, "ncounts" means "number of counts."

Any lowercase word that begins with an i or an n means that the name of an integer variable or constant is required in that position in the sequence. (Unless otherwise specified, integer means a one-word integer, the default INTEGER*2 data format.) The initial letter i is mnemonic for any integer; the initial letter n is mnemonic for "number of ...." For example, iflag in the sample sequence refers to an integer variable or constant; nvalues refers to an integer variable or constant containing the number of values.

Any lowercase word that begins with an r means that the name of a real variable or constant is required in that position in the sequence. (Unless otherwise specified, real refers to the default REAL*4 data format.) For example, rvalue in the sample sequence refers to a real variable.

Any lowercase word that begins with a letter other than i, n, or r means that the name of a subprogram is required at that point in the sequence. For example, call in the sample sequence refers to a subprogram name.

Any item in square brackets can be defaulted (left out of the sequence). The syntax always specifies exactly what portions of the sequence can be defaulted. For example, ",[iflag]" indicates that the comma is required regardless of whether iflag appears. The following call is valid with the argument iflag defaulted:

```
CALL SUBR(NCOUNT,SUM,,SUM2,MEANS)
```

When items in square brackets are defaulted from the rightmost end of the sequence, trailing commas are sometimes required by the syntax. However, for the sample sequence above,

```
use this form:  CALL SUBR(NCOUNT,SUM)
not this form:  CALL SUBR(NCOUNT,SUM,,,)
```

Any nested square brackets indicate more complex cases of the default syntax. The nesting of rvalue2 and call in the sample sequence above

means that rvalue2 and call cannot be defaulted independently. Either both rvalue2 and call can be defaulted (indicated by the outer brackets), or rvalue2 can be defaulted while call is specified (indicated by the inner brackets). However, if rvalue2 is specified, call must also be specified.

For example, the following calls are valid:

```
    CALL SUBR(NCOUNT,SUM,,SUM2,MEANS)
    CALL SUBR(NCOUNT,SUM,,,MEANS)
    CALL SUBR(NCOUNT,SUM,IFLAG)
```

These calls are not valid:

```
    CALL SUBR(NCOUNT,SUM,,SUM2,)
    CALL SUBR(NCOUNT,SUM,,SUM2)
```

An asterisk following the name of an argument is not part of the calling sequence. The asterisk convention marks those arguments that can be modified when the routine is called with the first argument omitted. This capability is referred to as parameter-adjustment mode. In the following example, the call to SUBR results in a change to the parameter iflag. According to the syntax, the second argument cannot be modified in parameter-adjustment mode. Therefore, the variable RATE is a dummy variable in a parameter-adjustment call, but according to the syntax, it cannot be defaulted.

```
    CALL SUBR(,RATE,IFLAG)
```

The character # following the name of an argument is not part of the calling sequence. The # convention indicates those arguments whose values can change during execution of the routine. Arguments marked with # must be variable names rather than constants. Under some circumstances, if you use a constant instead of a variable name, FORTRAN changes the value of the constant.

For example, suppose you use the following call:

```
    CALL SUBR(10,SUM,ITOT)
```

Suppose that SUBR returns 25 as the value of the first argument (nvalues). In some cases, SUBR would return 25 as the value of the constant 10. The constant 25 is then substituted for each subsequent appearance of the constant 10 in the program, with peculiar results for the program.

1.3.2 Notation Conventions

Any constant can be specified either as a decimal number or as an octal value. The system assumes that all FORTRAN constants are decimal unless they are marked as octal. You mark a constant as octal by prefixing the number with a double quote. For example, to select octal address 167772, use the constant "167772.

The manual refers to a bit with a value of 1 as being "on" or "set." A bit with value 0 is referred to as being "off" or "cleared."

The words "defaulted" and "omitted" differ in meaning when applied to subprogram arguments in this chapter. When the manual states that an argument can be defaulted, it means that the extensions routines assume a particular numeric value for the argument when it is left out of the argument list. When the manual states that an argument can be

omitted, it means that the extensions routines do not assume a value for the argument when it does not appear in the argument list. Instead, the extensions routines assume that the absence of the argument is a flag requesting some function other than that usually performed by the routine.

1.4 DESCRIPTION OF THE FORTRAN EXTENSIONS ROUTINES

The FORTRAN extensions routines are described in the following section. The routines are arranged alphabetically for convenient reference use. Section 1.1 contains functional groupings and descriptions of the routines so you can use those sections to determine which routines you need to solve your problem. Appendix A contains a reference summary of all the routines and their arguments.

The description of each routine contains the following information:

        Name of the routine
        Meaning of the name
        Functional operation of the routine
        Calling syntax
        Explanation of the arguments
        Error conditions
        Interactions with other routines
        Special cases or restrictions
        Examples

CLRD

1.4.1 CLRD (Clear Display) Routine

CLRD performs two functions on a single data array. It scales each element of the array by multiplying it by a scale factor supplied as one of the arguments. It calculates a horizontal (X-axis) spacing factor, which the display routines DIS and FSH use to position the data to fill the width of the screen.

CLRD is called as a subroutine.

        CALL CLRD(iarrayname#,npoints,ispace#,rscale)

The arguments are:

iarrayname   Name of the integer array containing the data to be scaled. Each array element is multiplied by rscale. Scaled values greater than 4095 (1023 for the AR11) are nondisplayable.

npoints   The number of elements in the array that are to be scaled. CLRD scales the first npoints array elements.

ispace   The X-axis spacing factor returned by CLRD. DIS and FSH use ispace to "stretch" the display horizontally so that the display fills the screen with evenly spaced points. CLRD calculates ispace as 4096/npoints (1024/npoints for the AR11). If npoints is greater than 4096 (or 1024 for the AR11), then ispace is set to 1.

                              NOTE

                 The internal machine representation of ispace is not in standard integer format. Therefore you cannot calculate your own value for ispace and you cannot use ispace in other calculations as if it were a normal integer value.

rscale   The scale factor used to multiply each element of iarrayname. The scale factor affects the Y-axis spacing by stretching or compressing the values to fill the screen as you require. If rscale is zero, then each element of the array is set to -1 (rather than zero) to indicate a nondisplayable value. Thus, you can initially make the whole array nondisplayable using CLRD, and then later selectively fill it with the points to be displayed.

Error conditions:

    SYNTAX ERROR
        The required four arguments are not present.

ARGUMENT ERROR
        The argument npoints is less than one.
        The argument rscale is negative.

Interactions:

    Both DIS and FSH require the spacing factor ispace calculated by
    CLRD.

    You can call CLRD to scale either or both of the data arrays for
    DXY and FXY.

Restrictions:

    None

Examples:

1. Set up an array ISCREEN that you will later fill with the data to
be displayed.   CLRD makes the array nondisplayable.   The value
returned in ISPACE is 1.

        DIMENSION ISCREEN(4096)
        CALL CLRD(ISCREEN,4096,ISPACE,0.0)

2. Suppose you have a data array IDATA of 250 points that you want to
prepare for display. The maximum value in the array is 1500, but you
would like the display to fill the screen vertically as well as
horizontally.   Calculate your scale factor by dividing the maximum
value you want by the maximum value you have.   In this case, the scale
factor might be 4095/1500 or approximately 2.67. For an AR11, the
scale factor might be 1023/1500, or roughly 0.67.

        CALL CLRD(IDATA,250,ISP,2.67)

If you want the data values scaled more exactly, use the following
call:

        CALL CLRD(IDATA,250,ISP,4095./1500.)

The value returned in ISP is equivalent to 16 (or 4 for the AR11).

CVSWG

1.4.2 CVSWG (Convert Switched Gain Value) Routine

CVSWG converts switched gain analog input values from packed integer format to real number format. CVSWG requires as input a value in the format returned by the RTS and IADC routines. The value contains the digitized value in the rightmost 12 bits for the AD11-K and LPS11 (10 bits for the AR11) and the gain code in bits 12 and 13. CVSWG separates this packed information into two values, the digitized value as a real number, and the gain code as an integer.

CVSWG is called as a function.

```
rvalue=CVSWG(ivalue[,igain#])
```

The arguments are:

ivalue	The integer value acquired using RTS or IADC. The real-number form of the digitized value is returned as CVSWG (the value of the function).
igain	The name of the variable in which the gain setting is returned. The value returned is in the range 1 to 4. The meanings of the gain codes appear in the descriptions of the routines IADC and RTS (see Sections 1.4.10 and 1.4.18, respectively).

Error conditions:

SYNTAX ERROR
    No arguments, or more than two arguments, are present.

Interaction:

The CVSWG routine normally converts data acquired by RTS or IADC.

Restrictions:

None

Examples:

1. You want to convert a value acquired (using IADC as a function) from channel 0 with a gain code of 2.

```
VAL=CVSWG(IADC(0,2))
```

2. You want to convert a value acquired by IADC from channel 2 with autogain. IGAIN contains the gain selected and used by IADC during the conversion. It is necessary to divide VAL by the gain value that IGAIN represents to restore VAL to its original scale.

```
VAL=CVSWG(IADC(2,0),IGAIN)
VAL=VAL/(2**(2*(IGAIN-1)))
```

The expression (2**(2*(IGAIN-1))) results in scale factors 1, 4, 16, and 64 for IGAIN values 1, 2, 3, and 4 (respectively).

DIS

1.4.3 DIS (Display Data) Routine

DIS initiates the continuous display of Y-axis data on a refresh-type scope (for example, the Tektronix 604).

DIS displays Y-axis data on the scope. The routine assumes that if X-axis values were provided, they would be equally spaced, increasing values. Logically, there are two X values for each Y value plotted. One specifies the X coordinate on the screen where the Y value appears, and the other specifies the array subscript where the Y value can be found. The X coordinate and array subscript for a single point usually have different values. You do not have to provide explicit X values. The X-coordinate value is controlled by an X-axis spacing factor, whereas the array subscript value is controlled by an indexing factor (see the ispace and increment argument descriptions).

DIS displays one data point on every interrupt from the scope control so that it displays points as quickly as the scope can receive them. DIS constantly plots points, starting again at the beginning of the array as soon as it reaches the end.

DIS is called as a subroutine.

        CALL DIS(iydata,ispace,npoints,istart,increment)

The arguments are:

iydata	The name of the integer array containing the scaled Y-axis data to be displayed. The Y values should be in the range 0 to 4095 (or 0 to 1023 for the AR11). (See the restriction on Y values later in this section.)
ispace	The X-axis spacing factor. The ispace argument controls the horizontal spacing (X coordinates) for the points plotted. You must call CLRD to calculate ispace before you call DIS. If ispace is 1 and increment is 1, then the X-axis coordinate for the display assumes values 0, 1, 2 ,...,(npoints - 1). If ispace is greater than 1, then the X-axis coordinate for the display assumes values 0, ispace*increment, 2*ispace*increment, ...,(npoints-1)*ispace*increment.
npoints	The number of points to display. The npoints argument must be greater than zero.
istart	The array subscript of the first point to be displayed from iydata. The istart argument must be a legal subscript within the bounds of the array iydata.
increment	The indexing factor for the array subscripts. DIS adds increment to the preceding subscript to obtain the subscript of the next point to display. The starting subscript is istart. If increment is 1, then the points displayed have subscript values istart through (istart + npoints - 1). If

increment is greater than 1, then the subscripts have values istart through istart + increment* (npoints - 1). No error message appears when the array subscript goes out of bounds.

Error conditions:

SYNTAX ERROR
The required five arguments are not present.

ARGUMENT ERROR
The number of points requested (npoints) is less than 1 or greater than 4096.
The subscript of the first point to be displayed (istart) was less than 1.
The increment specified was less than 1 or greater than 4095.

Interactions:

DIS requires the horizontal spacing factor (ispace) calculated by CLRD.

DIS displays the next data point on every interrupt from the scope control. A hardware interrupt occurs after each point displayed by DIS, telling DIS that the scope control is ready to receive the next data point. The time interval from the display of a point to the occurrence of an interrupt may be as brief as 22 microseconds. Therefore, the remaining machine resources are not sufficient to allow much data processing or computation. The only other routines that are guaranteed to be running are other interrupt handling routines (for example, the clock-interrupt service routine) and completion routines. Therefore, the most practical way to stop DIS displays is to call SDIS in a completion routine.

Restrictions:

The range of voltages accepted by some scopes is narrower than the range of voltages produced by the D/A converters. For example, full scale (0 to 4095) for the converter may represent -5.12 to 5.12 volts. However, the scope control may accept display voltages in the range -5 to +5 volts. Therefore, digital values 0 through 47 and 4048 through 4095 are not displayed on those scopes, and displays including those values hang off the edges of the screen. To avoid losing data with these scopes, arrange the arguments and data so that the first 47 screen positions correspond to nondisplayable data in the data array. Fill the data array with displayable values so that the displayable values have X coordinates in the range 48 through 4047. The scaled Y values should be in the range 48 through 4047.

Negative data values, or values greater than 4095 (1023 for the AR11) are nondisplayable. A point with a value greater than 4095 (1023) is treated as if its value were -1.

Each point is intensified every npoints interrupts. The amount of flicker in the display depends on the number of points and the persistence of the phosphor in the scope. Nondisplayable points are not ignored but are processed as if they were appearing on the screen. If flicker is a problem, first try to reduce the number of nondisplayable points being "displayed."

Examples:

1. There is a data array 1000 points long. The horizontal spacing factor ISPACE was calculated in a prior call to CLRD.

      (a) You want to display the last 800 points in the array.

```
CALL DIS(IDATA,ISPACE,800,201,1)
```

      (b) You want to display a data window of the middle 750 points, displaying every fifth point.

```
CALL DIS(IDATA,ISPACE,150,126,5)
```

2. The following example is a complete short program designed for a refresh-type scope. The example calls the routine SETR to control the duration of the display and to set up the call to the completion routine containing the command to stop the display.

```
    DIMENSION ISCREEN(4096)                 ;Array to be displayed.
    EXTERNAL STOPS                          ;Completion routine name
                                            ;declared EXTERNAL.
    CALL CLRD(ISCREEN,4096,ISPACE,0.0)      ;Make array
                                            ;nondisplayable and
                                            ;ispace=1.
    DO 10 I=48,4047,20                       ;DO limits chosen to
                                            ;fill screen.
10  ISCREEN(I)=I                            ;Put values in ISCREEN
                                            ;for diagonal line.
    NPTS=200                                ;Specify 200 points.
20  IFLG=0                                  ;Set up SETR flag.
    CALL SETR(5,0,1000.,IFLG,,STOPS)        ;Time for 10 sec, then
                                            ;call the completion
                                            ;routine STOPS.
    CALL DIS(ISCREEN,ISPACE,NPTS,48,20)     ;Display every 20th
                                            ;point in ISCREEN
                                            ;starting with the
                                            ;48th point.

    PAUSE                                   ;Wait for keyboard
                                            ;input.
    GO TO 20                                ;Display line again.
    END

    SUBROUTINE STOPS
    CALL SDIS                               ;Stop the display.
    RETURN
    END
```

1.4.4 DRS (Digital Read-in Sampling) Routine

DRS mediates repeated sampling from a digital input register (DR11-K or LPSDR-A) or from a memory location. The DRS routine extends your digital sampling capabilities beyond the single register reading that you can acquire using IDIR.

DRS collects multiple samples from a single input unit or memory location. Use multiple calls to DRS to sample from multiple units or memory locations.

DRS provides three basic capabilities.

1. Single-sweep sampling. Using DRS, you can acquire a short sequence of digital input register readings by specifying the parameters that direct DRS to stop after it fills an array with data.

2. Continuous sampling. Using DRS, you can acquire digital input data continuously. There are two classes of continuous sampling: finite continuous sampling, in which you define the total number of samples to be acquired in the call to DRS, and infinite continuous sampling, in which you do not specify the total number of samples in the call to DRS.

   You specify the parameters directing DRS to acquire data continuously, and either designate a completion routine or arrange your program code to empty the arrays periodically. You designate the completion routine in the SETR call if you are acquiring data on clock interrupts; you designate the completion routine directly from DRS if you are acquiring data on interrupts from the digital input interface. If you request infinite continuous sampling, you must call DRS again to stop sampling.

   You provide the completion routine to empty the arrays and dispose of the data as required by the problem. For example, you can write the data to secondary storage or reduce it to means or counts.

   Because the completion routine requires time to execute, the maximum continuous sampling rate (involving a completion routine) is lower than the maximum single-sweep sampling rate (not involving a completion routine).

3. End sampling. Call DRS in parameter-adjustment mode to stop infinite continuous sampling.

DRS is called as a subroutine. The number of arguments necessary depends on the capabilities you are requesting.

   Single-sweep sampling:

```
CALL DRS (iarrayname,iarraysize,,[nsamples],[isource],iunit,
        imask,[imode],iendflag#,nleft)
```

Continuous sampling:

(a) Clock-interrupt-driven continuous sampling

    CALL DRS (iarrayname,iarraysize,[nsubarrays],[nsamples],
         [isource],iunit,imask,[imode],iendflag#,nleft#)

(b) Event-driven continuous sampling

    CALL DRS (iarrayname,iarraysize,[nsubarrays],[nsamples],
         [isource],iunit,imask,[imode],iendflag#,nleft#[,
         [interval],complete])

End sampling:

Specify parameter-adjustment mode by omitting iarrayname. Only
imode can be altered. The values of isource and iunit specify
the register or location for which sampling is to stop. The rest
of the variable names are dummy arguments that should be the same
as the names in the original DRS call.

    CALL DRS (,iarraysize,,,[isource],iunit,imask,imode*,
         iendflag,nleft)

The arguments are:

iarrayname   The name of the integer array in which DRS places
                the input data. For continuous sampling, DRS
                treats the data array as a circular array (see
                Section 1.2.6).   If you omit iarrayname, the DRS
                call expects a negative value for imode to end
                sampling.

iarraysize   The length of iarrayname (in words).   The
                iarraysize argument specifies the amount of memory
                reserved by your program for input data.

nsubarrays   The number of array partitions in iarrayname. The
                default number of partitions is 1. The length of
                each partition is equal to the largest integer in
                (iarraysize/nsubarrays).   For   example,   with
                iarraysize = 512, and nsubarrays = 5, there are
                five partitions, each 100 words long, and 12
                unused words at the end of iarrayname.

                The value of nsubarrays must be less than or equal
                to   iarraysize.   With continuous double-word
                sampling, there must be at least two array
                partitions   (see   the   interval   argument
                description). Single-sweep sampling does   not
                require array partitions.

nsamples   The number of samples required. A single sample
                is either a single data word or a pair of words,
                depending on the mode specified. The default
                value of nsamples is iarraysize.

                With single-sweep sampling, set nsamples equal to
                iarraysize for single-word sampling modes; set
                nsamples equal to half the value of iarraysize for
                double-word sampling modes.

                With continuous sampling, you assign the value to
                nsamples depending on how you want to stop the

sampling process. For finite continuous sampling, set nsamples equal to the number of samples you want. In this case,

nsamples > iarraysize

For infinite continuous sampling, set nsamples equal to any negative number.

nsamples < 0

DRS then reads data until you stop sampling by calling DRS again in parameter-adjustment mode with a negative value for imode.

isource
The flag indicating whether to read a digital input register or a memory location. The default value of isource is zero.

Value	Meaning
zero	Read the digital input register specified by iunit. In the process of reading the register, DRS clears those bits (and only those bits) that it finds set.
nonzero	Read the memory location whose address appears in iunit.

iunit
The location of the data to be read.

Value	Meaning
0 through 7 or 0 through 8	Logical unit numbers for the digital input registers. The 0-through-7 values select one of up to eight DR11-K digital I/O interfaces. If the system configuration includes an LPS11 with LPSDR-A, then 0 selects the LPSDR-A, and the 1-through-8 values select one of up to eight DR11-K digital I/O interfaces. (The value of isource must be zero.)
0,2,...,"177776	Memory address selected. The address must be an even number. (The value of isource must be nonzero.)

imask
The value used to mask the input value. DRS performs a logical AND using imask and the input value, and stores the result in the data array, iarrayname. Request a mask with all bits on either as -1 (decimal) or as "177777 (octal).

imode
For single-word sampling, imode specifies the data format of the value to be read and the source of the interrupts driving the sampling.

For double-word sampling, imode specifies the data format of the value to be read, the source of the interrupts driving the sampling, and the contents of the second word.

For parameter adjustment, imode specifies the stop code.

The default value of imode is zero.

Single-word sampling.
In single-word sampling modes, DRS reads the contents of the digital register or the memory location into the data array. The single-word sampling codes are:

Value	Meaning
0	Read unsigned BCD data from the register or memory location on clock interrupt.
1	Read binary data from the register or memory location on clock interrupt.
4	Read unsigned BCD data from the register or memory location on a digital event interrupt.
5	Read binary data from the register or memory location on a digital event interrupt.

Double-word sampling.
There are two kinds of double-word sampling: double-word sampling with bit location, and double word sampling with time stamping. In both, the first word of the pair contains the register or memory value.

In double-word sampling with bit location, the second word of the pair contains the bit position of the rightmost nonzero bit in the nonmasked portion of the input value. That is, DRS performs a logical AND between the input value and the complement of the mask, and then scans right to left for a nonzero bit. It assigns the bit position (0 to 15) as the value of the second word. If no bits are on, the second word of the pair is set to -1. You request double-word sampling with bit location by adding 2 to any of the single-word sampling codes to obtain codes 2, 3, 6, and 7.

In double-word sampling with time stamping, the second word of the pair contains the software clock reading at the time of the digital event interrupt. The clock value does not change. The time-stamping option is available only with digital event interrupts. You use double-word sampling with time stamping to obtain information concerning what happened (the digital input register) and when it happened (the software-clock

reading). You request double-word sampling with time stamping by adding 8 to modes 4 and 5 to obtain codes 12 and 13.

The double-word sampling codes are:

Value	Meaning
2	Double-word sampling with bit location. Read and convert unsigned BCD data on clock interrupt.
3	Double-word sampling with bit location. Read binary data on clock interrupt.
6	Double-word sampling with bit location. Read and convert unsigned BCD data on digital event interrupt.
7	Double-word sampling with bit location. Read binary data on digital event interrupt.
12	Double-word sampling with time stamping. Read and convert unsigned BCD data on digital event interrupt.
13	Double-word sampling with time stamping. Read binary data on digital event interrupt.

End sampling.
Each parameter-adjustment call to DRS stops sampling from one input unit or memory location. If you are sampling from more than one unit or location, use one parameter-adjustment call for each unit to be stopped. (Request parameter-adjustment mode by omitting the iarrayname argument.)

The stop code values are:

Value	Meaning
-4	Finish sampling for the current array partition. Allow queued completion routine requests to run, but do not queue any more requests.
-3	Finish sampling for the current array partition. Do not honor any queued completion routine requests.
-2	Stop reading data. Allow queued completion routine requests to run, but do not queue any more requests.
-1	Stop reading data. Do not honor any queued completion routine requests.

iendflag   The completion and/or error flag. You must set
                iendflag to zero before you call DRS. DRS
                increments iendflag either when all samples
                requested have been acquired or when sampling has
                been stopped by a stop code (negative value for
                imode). If a data overrun or other error occurs,
                DRS sets iendflag to a negative value.

nleft   The array partition flag. DRS initializes nleft
                with the number of array partitions specified by
                nsubarrays. DRS decrements nleft by 1 each time
                an array partition becomes full. Thus nleft
                contains the number of partitions currently
                available to DRS. The nleft argument is
                meaningful only with continuous sampling.
                Remember to increment nleft when you remove data
                from a full array partition so that the partition
                can be reused.

                If nleft becomes zero before the number of samples
                requested in nsamples has been acquired, a data
                overrun has occurred. DRS then sets iendflag to
                indicate an error.

interval   The number of interrupts between calls to the
                completion routine. DRS initializes an internal
                variable to interval and decrements it after each
                interrupt. When the value is zero, DRS
                reinitializes the internal variable and causes the
                completion routine to be called. Therefore, the
                completion routine is executed whenever interval
                interrupts have occurred.

                The default value for interval is the length of
                the array partition, iarraysize/nsubarrays. With
                single-word sampling, the default value for
                interval causes the completion routine to be
                executed as each array partition is filled. With
                double-word sampling, the default value for
                interval causes the completion routine to be
                executed after two array partitions have been
                filled.

complete   Completion-routine name. The completion routine
                that you provide is executed after interval
                interrupts have occurred. There is no default
                completion routine name. If you omit the
                completion routine name in digital event driven
                modes, DRS does not call a completion routine. In
                clock-driven modes, you must omit the completion
                routine argument and instead specify any necessary
                completion routine in the call to the clock
                routine, SETR.

Error conditions:

    SYNTAX ERROR
        There are fewer than 10, or more than 12, arguments present.

    ARGUMENT ERROR
        The digital input register (iunit) specified does not exist.
        The memory address (iunit) specified is an odd number.
        The mode requested (imode) is negative but the call is not
        in parameter-adjustment mode.

DEVICE CONFLICT
    The digital input register is in use.

Interactions:

In clock-driven sampling modes, you must designate any completion-routine name in the SETR call.

Restrictions:

DRS allows you to sample as many as eight DR11-K digital input registers (and the LPSDR-A, if present) simultaneously. Call DRS once for each unit you want to read, specifying a different array name for each unit. Each unit can run in either clock-driven or event-driven mode. You can designate a unique completion routine for each of the units running in event-driven mode.

Examples:

1. You want to read digital input register 5 once each second for one minute. DRS reads one word of binary data into the array MIN on each clock overflow interrupt.

```
DIMENSION MIN(60)                        ;Set up data array.
IEND=0                                   ;End flag for DRS.
IENDC=0                                  ;End flag for clock.
CALL DRS(MIN,60,,,,5,-1,1,IEND,IFULL)    ;Request sampling.
CALL SETR(5,1,100.,IENDC)                ;Start clock at 1 Hz.
CALL LWAIT(IEND,0)                       ;Wait for 60 samples.
CALL SETR(-1,,,)                         ;Stop the clock.
        .
        .
        .
```

2. You want to read 60 samples of BCD data from digital input register 3 on digital event interrupts. You are interested only in the high-order byte, so you mask off the low-order byte.

```
DIMENSION MIN(60)                        ;Set up data array.
IEND=0                                   ;End flag for DRS.
CALL DRS(MIN,60,,,,3,"177400,4,IEND,IFULL)
                                         ;Request sampling.
CALL LWAIT(IEND,0)                       ;Wait for 60 samples.
        .
        .
        .
```

3. You want to read the status register for DR11-K unit 0 and execute a completion routine on every digital event interrupt. You request double-word sampling with time stamping using mode 13. The example appears twice, once with completion routines, and once without completion routines.

(a) This example uses a completion routine. The example assumes that the interrupts occur at a rate slow enough to allow the completion routine to execute fully between interrupts.

```
DIMENSION IREG(4)                        ;Set up array.
EXTERNAL COMPL                           ;Label completion routine
                                         ;EXTERNAL.
IEND=0                                   ;End flag for DRS.
```

```
        CALL DRS(IREG,4,2,-1,1,"167770,-1,13,IEND,IFUL,1,COMPL)
                                              ;Request sampling.

          .
          .
          .

        SUBROUTINE COMPL                      ;Completion routine.
        for example,
          issue control instructions
          acquire or process data
          compute
          examine conditions to decide if done
        IF (IDONE.EQ.1) GO TO 10              ;Go to 10 if finished.
        IFUL=2                                ;Reset available space
                                              ;pointer.

        RETURN                                ;Return if more to do.
10      CALL DRS(,4,,,1,"167770,-1,-1,IEND,IFUL)
                                              ;Call DRS with stop
                                              ;code=-1.

        RETURN
```

(b) You can achieve the same function without completion routines.

```
        DIMENSION IREG(4)                     ;Define data destination.
        IEND=0                                ;End flag for DRS.
        CALL DRS(IREG,4,2,-1,1,"167770,-1,13,IEND,IFUL)
                                              ;Start sampling.
10      CALL LWAIT(IFUL,2)                    ;Wait for an interrupt.
          .
          .
          .

        do whatever is required, for example
          issue control instructions
          acquire or process data
          compute
          examine conditions to decide if finished
          go to 20 if finished
          .
          .
          .

        IFUL=2                                ;Reset available space
                                              ;pointer.
        GO TO 10                              ;Go wait for next event.
20      CALL DRS(,4,,,1,"167770,-1,-1,IEND,IFUL)
                                              ;Stop DRS with imode=-1.
          .
          .
          .
```

DXY

1.4.5 DXY (Display X-Y Data Pairs) Routine

DXY continuously displays X-Y data pairs contained in two data arrays.
The continuous display is used for refresh-type scopes (for example,
Tektronix 604). DXY plots correlational data, or any data in which
the Y values are not associated with equally spaced, ordered X values.

DXY displays one data point on each interrupt from the scope control,
so that it displays points as quickly as the scope can receive them.
DXY constantly plots data points, starting again at the beginning of
the arrays after it has displayed the last point in the arrays.

The data to be displayed must be in two arrays. The X-axis data array
contains the horizontal displacement of each point to be plotted. The
Y-axis data array contains the vertical displacement of each point to
be plotted. The displacements are relative to (0,0) in the lower left
corner of the screen. The X-Y pairs are in corresponding positions in
the two arrays. That is, for the fifth pair to be plotted, the X
value is the fifth element of the X array, and the Y value is the
fifth element of the Y array.

The data values must be in the range that can be plotted on the scope,
that is, 0 to 4095 (0 to 1023 for the AR11). You can expand or
compress the range of the X and Y data arrays independently by using
CLRD to scale them. You can also transform or scale the data arrays
elsewhere in the program with FORTRAN manipulations.

DXY is called as a subroutine.

        CALL DXY(ixdata,iydata,npoints,istart,increment)

The arguments are:

ixdata	The name of the integer array containing the X-axis values for the X-Y pairs to be displayed.
iydata	The name of the integer array containing the Y-axis values for the X-Y pairs to be displayed.
npoints	The number of points (X-Y pairs) to be plotted.
istart	The array subscript of the first pair of points to be plotted. The istart argument must be a legal subscript within the bounds of the arrays ixdata and iydata.
increment	The indexing factor for the array subscripts. DXY adds increment to the preceding subscript to obtain the subscript of the next pair of values to be displayed. If increment is 1, then the pairs displayed have subscript values istart through (istart + npoints - 1). If increment is greater than 1, then the subscripts have values istart through (istart + increment*(npoints - 1)). The increment argument must be greater than or equal to 1. No error message appears when the array subscript goes out of bounds.

Error conditions:

SYNTAX ERROR
The required five arguments are not present.

ARGUMENT ERROR
The number of pairs of points to display (npoints) is less than 1 or greater than 4096.
The array subscript of the first pair to be displayed (istart) is less than 1.
The increment requested is greater than 4096.

Interaction:

DXY displays the next data point on each interrupt from the scope control. A scope-control interrupt occurs after each point displayed, telling DXY that the scope control is ready to receive the next data point. The time interval from the display command to the occurrence of an interrupt may be as brief as 22 microseconds. Therefore, the remaining machine resources are not sufficient to allow much data processing or computation. The only routines that are guaranteed to be running are other interrupt handling routines (for example, the clock or keyboard interrupt service routines) and completion routines. Therefore, the most practical way to stop a DXY display is to call SDIS from a completion routine.

Restrictions:

The range of voltages accepted by some scopes is narrower than the range of voltages produced by the D/A converters. For example, full scale (0 to 4095) for the converter may represent -5.12 to +5.12 volts. However, the scope control may accept voltages in the range -5 to +5 volts. Therefore, digital values 0 through 47 and 4048 through 4095 are not displayed on those scopes, and displays including those values hang off the edges of the screen. To avoid losing data with these scopes, arrange the arguments and data so that the first 47 screen positions correspond to nondisplayable data in the data array. Scale both the X and Y arrays to have displayable values in the range 48 through 4047.

Data values greater than 4095 (1023 for the AR11) are nondisplayable. A point with a value greater than 4095 (1023) is treated as if its value were -1.

Each point is intensified every npoints interrupts. The amount of flicker in the display depends on the number of points and the persistence of the phosphor in the scope. Nondisplayable points are not ignored but are processed as if they were appearing on the screen. If flicker is a problem, first try to reduce the number of nondisplayable points being "displayed."

Example:

1. The example draws a horizontal line and a vertical line to divide the screen into quadrants. The display is stopped by SDIS in a completion routine that is designated in the call to SETR and executed after approximately one minute.

```
       EXTERNAL STOPS
       INTEGER XDATA(4096),YDATA(4096)
       CALL CLRD(XDATA,4096,IDUM,0.0)          ;Clear X array.
       CALL CLRD(YDATA,4096,IDUM,0.0)          ;Clear Y array.
       DO 10 I=48,4047,50                      ;Make horizontal line.
       XDATA(I)=I
10     YDATA(I)=2048
       DO 20 I=73,4047,50                      ;Make vertical line.
       XDATA(I)=2048
20     YDATA(I)=I
       NPTS=160                                ;Number of points to
                                               ;display.
30     IFLAG=0                                 ;End flag for SETR.
       CALL SETR(5,0,6000.,IFLAG,,STOPS)       ;Time 1 minute; then
                                               ;execute completion
                                               ;routine STOPS.
       CALL DXY(XDATA,YDATA,NPTS,48,25)        ;Display the arrays.
       PAUSE                                   ;Wait for keyboard input.
       GO TO 30                                ;Go start display again.
       END

       SUBROUTINE STOPS                        ;Completion routine.
       CALL SDIS                               ;Stop the display.
       RETURN
       END
```

FLT16

1.4.6 FLT16 (16-bit Floating-Point Conversion) Routine

FLT16 converts a 16-bit unsigned binary integer to real-number format. A 16-bit unsigned binary integer (with octal values 0 to 177777) has decimal values of 0 to 65535. (These values are in contrast to most integers which are stored in two's complement binary and can assume decimal values -32768 through 32767.) You must use FLT16 for unsigned binary values instead of the FORTRAN function FLOAT because FLOAT converts two's complement binary, not unsigned binary.

FLT16 is called as a function.

        rvalue = FLT16(ivalue)

The arguments are:

        rvalue   The real-number form of the integer argument.

        ivalue   The unsigned binary integer to be converted to a
                        real value.

Error conditions:

        SYNTAX ERROR
            There is not exactly one argument present.

Interaction:

        Unsigned binary values are acquired by IDIR, DRS, and IDOR.

Restrictions:

        None

Example:

1. Consider the unsigned binary value IBINRY:

        1 111 111 111 111 111

If

        VALUE=FLT16(IBINRY)

then VALUE contains the real-number form of 65535.

If

        VALUE=FLOAT(IBINARY)

then VALUE contains the real-number form of -1.

2. Consider the unsigned binary value IBIN:

        1 000 000 000 000 000

If

        VAL=FLT16(IBIN)

then VAL contains the real-number form of 32768.

If

        VAL=FLOAT(IBIN)

then VAL contains the real-number form of -32768.

3. You want to determine the time interval between two events.

```
DIMENSION IEVENT(4)                      ;Set up data array.
IFLG=0                                   ;End flag for DRS.
ISFLG=0                                  ;End flag for SETR.
CALL DRS(IEVENT,4,,2,,1,-1,13,IFLG,NLEFT)
                                         ;Read register 1, double
                                         ;word sampling with time
                                         ;stamping in event-driven
                                         ;mode.
CALL SETR(4,1,1.,ISFLG)                  ;SETR runs the software
                                         ;clock in msec.
CALL LWAIT(IFLG,0)                       ;Wait for end of
                                         ;sampling.
TIME=FLT16(IEVENT(4))-FLT16(IEVENT(2))   ;Interval in msec.
    .
    .                                    ;Go on to examine the
    .                                    ;digital registers.
```

FSH

1.4.7 FSH (Flash) Routine

FSH displays all points in a data array once, that is, "flashes" them on the screen. However, you can use FSH for continuous displays if you have a storage scope (for example, Tektronix 603), or if you use a completion routine to refresh the display for refresh-type scopes (for example, Tektronix 604).

FSH displays Y-axis data on the scope. The routine assumes that if X-axis values were provided, they would be equally spaced increasing values. Logically, there are two X values for each Y value plotted. One specifies the X coordinate on the screen where the Y value appears, and the other specifies the array subscript where the Y value can be found. The X coordinate and array subscript for a single point usually have different values. You do not have to provide explicit X values. The X-coordinate value is controlled by an X-axis spacing factor, whereas the array subscript value is controlled by an indexing factor (see the ispace and increment argument descriptions).

FSH is called as a subroutine.

        CALL FSH(iydata,ispace,npoints,istart,increment)

The arguments are:

iydata	The name of the integer array containing the scaled Y-axis data to be displayed.
ispace	The X-axis spacing factor. You must call CLRD to calculate ispace before you call FSH. If ispace is 1 and increment is 1, then the X-axis coordinate for the display assumes values $0,1,2,...,$(npoints -1). If ispace is greater than 1, the X-axis coordinate for the display assumes values 0, ispace*increment, 2*ispace*increment,..., (npoints $- 1$)*ispace*increment.
npoints	The number of points to display. The npoints argument must be greater than zero.
istart	The array subscript of the first point to be displayed from iydata. The istart value must be a legal subscript within the bounds of the array iydata.
increment	The indexing factor for the array subscripts. FSH adds increment to the preceding subscript to obtain the subscript of the next point to be displayed. The starting subscript is istart. If increment is 1, then the points displayed have subscript values istart through (istart + npoints $- 1$). If increment is greater than 1, then the subscripts have values istart through istart + increment*(npoints $- 1$). No error message appears if the array subscript goes out of bounds.

Error conditions:

> SYNTAX ERROR
>> There are not exactly five arguments present.
>
> ARGUMENT ERROR
>> The number of points to be displayed (npoints) is less than zero or greater than 4096.
>> The increment for the array subscript is less than 1 or greater than 4096.
>> The subscript of the first point to be displayed (ifirst) is less than 1.

Interactions:

> A CLRD call calculating the display spacing factor (ispace) must have preceded the FSH call.

> You can obtain a continuous display on a refresh-type scope by calling FSH from a completion routine. FSH then executes whenever your completion routine executes. The rate of completion-routine execution (the refresh rate) depends on the parameters you specify when you initialize the clock (in SETR or the SYSLIB routine ITIMER).

> Unlike continuous displays with DIS and DXY, the completion routine method of obtaining continuous displays may not use all the machine resources. The amount of time remaining depends on the number of points in the display and on how often the completion routine executes. Your FORTRAN program might be able to process data or perform computations while the display is on the screen.

Restrictions:

> The range of voltages accepted by some scopes is narrower than the range of voltages produced by the D/A converters. For example, full scale (0 to 4095) for the converter may represent -5.12 to 5.12 volts. However, the scope control may accept voltages in the range -5 to +5 volts. Therefore, digital values 0 through 47 and 4048 through 4095 are not displayed on those scopes, and displays including those values hang off the edges of the screen. To avoid losing data with these scopes, arrange the arguments and data so that the first 47 screen positions correspond to nondisplayable data in the data array. Fill the array with displayable values so that the displayable values have X coordinates in the range 48 through 4047. The scaled Y values should be in the range 48 through 4047.

> Data values greater than 4095 (1023 for the AR11) are nondisplayable. A point with a value greater than 4095 (1023) is treated as if its value were -1.

> Each point is intensified every npoints interrupts. The amount of flicker in the display depends on the number of points and the persistence of the phosphor in the scope. Nondisplayable points are not ignored but are processed as if they were appearing on the screen. If flicker is a problem, first try to reduce the number of nondisplayable points being "displayed."

Examples:

1. The following example is a complete short program designed for a storage scope. The example erases the screen and plots a diagonal

line of points that remain on the screen until you press a key, at which time it erases the screen and replots the line. The only way to stop the program is to enter a CTRL/C. After you enter a CTRL/C, the scope remains in storage mode. If you want to clear the screen, type INIT after the monitor dot for RT-11 SJ, or reboot the system for RT-11 FB.

```
        INTEGER SCREEN(4096)                ;Set up array space.
        CALL ERASE                          ;Be sure screen is blank.
        CALL CLRD(SCREEN,4096,IDELT,0.0)    ;Make SCREEN nondisplayable.
        DO 10 I=48,4047,3                   ;DO limits chosen to fill
                                            ;screen.
10      SCREEN(I)=I                         ;Create points for diagonal
                                            ;line.
        NPT=1333                            ;No. of points to display.
        TYPE 20,NPT                         ;Print message.
20      FORMAT(/'  NO. OF POINTS DISPLAYED =',I5)
30      CALL FSH(SCREEN,IDELT,NPT,48,3)     ;Display array.
        PAUSE                               ;Wait for keyboard entry.
        CALL ERASE                          ;Erase screen.
        GO TO 30                            ;Go display line again.
        END

        SUBROUTINE ERASE                    ;You write this routine.
        IFLAG=0                             ;Initialize flag for SETR.
        CALL IDOR(1,"170446,"6,"6)          ;Turn on storage mode and
                                            ;non-erase flag.
        CALL IDOR(1,"170446,2,0)            ;Clear erase bit thus starting
                                            ;erase cycle.
        CALL SETR(4,0,250.,IFLAG)           ;Start 250 msec interval to
                                            ;wait for settling time.
        CALL LWAIT(IFLAG,0)                 ;Wait for end of interval.
        RETURN
        END
```

2. The following example shows the use of FSH for continuous displays on a refresh-type scope (for example, Tektronix 604). FSH is called from within a completion routine which is executed on interrupts from the programmable clock.

```
        EXTERNAL SHOW                       ;Define the completion
                                            ;routine.
        COMMON IYDATA(4096),N,ISP           ;Make array available to
                                            ;completion routine.
        CALL CLRD(IYDATA,4096,ISP,0.)       ;Make IYDATA nondisplayable.
        DO 10 I=48,4047,20                  ;DO limits chosen to fill
                                            ;screen.
10      IYDATA(I)=I                         ;Fill array with points.
        N=200                               ;No. of points to display.
        TIME=100.                           ;Time interval between calls
                                            ;to the completion routine (in
                                            ;msec).
        IFLAG=0                             ;Flag for SETR.
        CALL SETR(4,1,TIME,IFLAG,,SHOW)     ;Get clock going and prepare
                                            ;for completion routine.
        PAUSE                               ;Wait for keyboard entry.
        CALL SETR(-1,,,)                    ;Stop the clock.
        END

        SUBROUTINE SHOW                     ;Completion routine.
        COMMON IYDATA(4096),N,ISP
        CALL FSH(IYDATA,ISP,N,48,20)        ;Display the array.
        RETURN
        END
```

FXY

1.4.8 FXY (Flash X-Y Data Pairs) Routine

FXY displays X-Y data pairs once (that is, "flashes" them on the screen). However, you can use FXY for continuous displays if you have a storage scope (for example, Tektronix 603), or if you use a completion routine to refresh the display for refresh-type scopes (for example, Tektronix 604).

You can use FXY to plot any data in which the Y values are not associated with equally spaced, increasing X values (for example, scatter-plot data).

FXY displays X-Y data pairs contained in two data arrays. The X-axis data array contains the horizontal displacement of each point to be plotted. The Y-axis data array contains the vertical displacement of each point to be plotted. The displacements are relative to (0,0) in the lower left corner of the screen. The X-Y pairs are in corresponding positions in the two arrays. That is, for the fifth pair to be plotted, the X value is the fifth element of the X array, and the Y value is the fifth element of the Y array.

The data values must be in the range that can be plotted on the scope, that is, 0 to 4095 (0 to 1023 for the AR11). You can expand or compress the range of the X and Y data arrays independently by using CLRD to scale them. You can also transform or scale the data arrays elsewhere in the program with FORTRAN manipulations.

FXY is called as a subroutine.

        CALL FXY(ixdata,iydata,npoints,istart,increment)

The arguments are:

ixdata
: The name of the integer array containing the X-axis values for the X-Y pairs to be displayed.

iydata
: The name of the integer array containing the Y-axis values for the X-Y pairs to be displayed.

npoints
: The number of points (X-Y pairs) to be plotted.

istart
: The array subscript of the first pair of points to be plotted. The istart argument must be a legal subscript within the bounds of the arrays ixdata and iydata.

increment
: The indexing factor for the array subscripts. FXY adds increment to the preceding subscript to obtain the subscript of the next pair of values to display. If increment is 1, then the pairs displayed have subscript values istart through (istart + npoints - 1). If increment is greater than 1, then the subscripts have values istart through (istart + increment*(npoints - 1). Increment must be greater than or equal to 1. No error message appears if the array subscript goes out of bounds.

Error conditions:

SYNTAX ERROR
There are not exactly five arguments present.

ARGUMENT ERROR
The number of pairs of points to display (npoints) is less than 1 or greater than 4096.
The array subscript of the first pair to be displayed (istart) is less than 1.
The increment requested is greater than 4096.

Interactions:

You can obtain a continuous display on a refresh-type scope by calling FXY from a completion routine. FXY then executes whenever your completion routine executes. The rate of completion-routine execution (the refresh rate) depends on the parameters you specify when you initialize the clock (in SETR or the SYSLIB routine ITIMER).

Unlike continuous displays with DIS and DXY, the completion-routine method of obtaining continuous displays may not use all the machine resources. The amount of time remaining depends on the number of points in the display and on how often the completion routine executes. Your FORTRAN program may be able to process data or perform computations while the display is on the screen.

Restrictions:

The range of voltages accepted by some scopes is narrower than the range of voltages produced by the D/A converters. For example, full scale (0 to 4095) for the converter may represent -5.12 to +5.12 volts. However, the scope control may accept voltages in the range -5 to +5 volts. Therefore, digital values 0 through 47 and 4048 through 4095 cannot be displayed on those scopes, and displays including those values hang off the edges of the screen. To avoid losing data with these scopes, arrange the arguments and data so that the first 47 screen positions correspond to nondisplayable data in the data array. Scale both the X and Y arrays to have displayable values in the range 48 through 4047.

Data values greater than 4095 (1023 for the AR11) are nondisplayable. A point with a value greater than 4095 (1023) is treated as if its value were -1.

Each point is intensified every npoints interrupts. The amount of flicker in the display depends on the number of points and the persistence of the phosphor in the scope. Nondisplayable points are not ignored but are processed as if they were appearing on the screen. If flicker is a problem, first try to reduce the number of nondisplayable points being "displayed."

Examples:

1. The following example shows the use of FXY for continuous displays with a refresh-type scope. FXY is called from within a completion routine which executes on interrupts from the clock counter. The example program draws a horizontal line and a vertical line to divide the screen into quadrants.

```
        COMMON IXDATA(4096),IYDATA(4096),N        ;Make arrays available to
                                                  ;completion routine.
        EXTERNAL SHOW                             ;Define completion routine
                                                  ;external.
        CALL CLRD(IXDATA,4096,ISP,0.)             ;Empty X array.
        CALL CLRD(IYDATA,4096,ISP,0.)             ;Empty Y array.
        DO 10 I=48,4047,50                        ;Create horizontal line.
        IXDATA(I)=I
10      IYDATA(I)=2048
        DO 20 I=73,4047,50                         ;Create vertical line.
        IXDATA(I)=2048
20      IYDATA(I)=I
        N=160                                     ;No. of points to display.
        IFLAG=0                                   ;End flag for SETR.
        TIME=100.                                 ;Refresh rate is 100 clock
                                                  ;ticks.
        CALL SETR(4,1,TIME,IFLAG,,SHOW)           ;Start clock in msec,
                                                  ;designate completion
                                                  ;routine SHOW.
        PAUSE                                     ;Wait for keyboard entry.
        CALL SETR(-1,,,)                          ;Stop the clock.
        END

        SUBROUTINE SHOW                           ;Completion routine.
        COMMON IXDATA(4096),IYDATA(4096),N        ;Data arrays from main
                                                  ;program.
        CALL FXY(IXDATA,IYDATA,N,48,25)           ;Display arrays.
        RETURN
        END
```

2.   The following example shows the use of FXY to display data arrays
on a storage scope.   The data are collected from analog channel 2,
with sampling started by an external event (for example, a stimulus
pulse). You can compare the two sweeps of data visually by looking at
the scatter plot.

```
        DIMENSION IDATA1(100),IDATA2(100)         ;Define the data arrays.
10      ICLCK=0                                   ;End flag for SETR.
        IEND=0                                    ;End flag for RTS.
        NPT=100                                   ;Length of data sweep.
        CALL SETR(5,5,1.,ICLCK)                   ;Request external start
                                                  ;with clock at 100Hz.
        CALL RTS(IDATA1,NPT,,,2,,1,2,IEND,IDUMMY)
                                                  ;Request single sweep of
                                                  ;100 points on clock
                                                  ;interrupts.
        CALL LWAIT(IEND,0)                        ;Wait for samples.
        CALL SETR(-1,,,)                          ;Stop the clock.
        ICLCK=0                                   ;Reset clock flag.
        CALL SETR(5,5,1.,ICLCK)                   ;Request clock again.
        IEND=0                                    ;Reset RTS end flag.
        CALL RTS(IDATA2,NPT,,,2,,1,2.IEND,IDUMMY)
                                                  ;Request 2nd sweep.
        CALL LWAIT(IEND,0)                        ;Wait for 2nd sample.
        CALL SETR(-1,,,)                          ;Stop the clock.
        CALL ERASE                                ;Erase the screen.
        CALL FXY(IDATA1,IDATA2,NPT,1,1)           ;Display scatter plot.
        PAUSE                                     ;Type character to finish.
        CALL ERASE                                ;Clear screen.
        GO TO 10                                  ;Repeat whole program.
        END

        SUBROUTINE ERASE                          ;Routine to clear screen.
        IFLAG=0                                   ;End flag for SETR.
```

```
CALL IDOR(1,"170446,6,6)              ;Turn on storage mode and
                                      ;nonerase flag.
CALL IDOR(1,"170446,2,0)              ;Clear erase bit to start
                                      ;erase cycle.
CALL SETR(4,0,250.,IFLAG)             ;Start 250 msec interval
                                      ;to wait for settling time.
CALL LWAIT(IFLAG,0)                   ;Wait for end of interval.
RETURN
END
```

HIST

1.4.9 HIST (Time Interval Sampling Technique) Routine

HIST reads the current value of the programmable clock counter on each clock interrupt and stores the time readings in a data array. Depending on the clock mode, you can time the intervals between successive events (modes 3 and 7) or you can determine the latencies of a series of events relative to a single event (modes 2 and 6).

HIST is called as a subroutine.

    CALL HIST(iarrayname,iarraysize,nsamples,iendflag#,nleft#)

The arguments are:

iarrayname   The name of the integer array in which HIST places the time data.

iarraysize   The length of iarrayname. The iarraysize value must be greater than or equal to 1.

nsamples   The number of time interval values to collect. Normally, HIST continues sampling until all samples requested have been read. However, you can stop sampling before all samples have been collected by calling HIST again with nsamples equal to zero. (In this case, the arguments other than nsamples should be the same as in the previous call to HIST.)

The nsamples value may be greater than iarraysize, in which case HIST treats the data array as a circular array (see Section 1.2.6). You must write your program to empty the array periodically or the data will be overwritten. HIST itself does not provide any completion-routine option, but you can designate a completion routine in the call to SETR.

iendflag   The completion and error flag. You must set iendflag to zero before you call HIST. HIST increments iendflag when all samples requested have been collected. If a data overrun or other error occurs, then HIST sets iendflag to -1.

nleft   The flag containing the number of array elements available for data before data overrun. HIST initializes nleft to the value in iarraysize. HIST decrements nleft after each data point is collected. Data overrun occurs when nleft becomes zero before all the samples have been collected. If you are removing data from iarrayname while HIST is running, you must increment nleft once for each point you remove.

Error conditions:

    SYNTAX ERROR
        There are not exactly five arguments present.

ARGUMENT ERROR
        The array size requested is less than 1.
        The number of samples requested is less than zero.

Interactions:

    HIST runs in close conjunction with SETR. You must call SETR to
    set the clock in an external-event timing mode (mode 2 or 3) and
    to set the clock rate. You connect the sensor for your external
    events to Schmitt trigger #2 of the clock counter. The
    occurrence of an external event fires Schmitt trigger #2, causing
    the time of occurrence to be recorded. If you want to start the
    clock with an external-event pulse, connect that event to Schmitt
    trigger #1 and specify one of the external-start clock modes for
    external-event timing (mode 6 or 7).

    HIST is driven by interrupts from the clock counter. Therefore,
    a completion routine designated in SETR to service clock Schmitt
    trigger #2 interrupts can process the data acquired by HIST. For
    example, you can tally the data for time interval histograms.
    You should specify an interrupt interval in SETR that is half of
    iarraysize so that the completion routine executes when the array
    is half full.

Restrictions:

    HIST does not work for the AR11.

    The maximum possible time interval that can be recorded depends
    on the clock counter rate. The maximum count at any clock
    counter rate is 65535. The faster the rate, the shorter the
    actual interval; the slower the rate, the longer the actual
    interval.

    You cannot detect clock overflow. When the clock counter
    increments from 65535 to 0 and continues to count, the clock
    counter does not give you any signal.

Examples:

1. The following example collects 40 time-interval values. The clock
counter is running in external event timing from zero base mode (mode
3). When an event occurs on Schmitt trigger #2, HIST places the
current clock counter value in the array ITIMES and the clock counter
automatically resets to zero.

```
    DIMENSION ITIMES(40)                    ;Define data array.
    IFLAG1=0                                ;HIST completion flag.
    IFLAG2=0                                ;SETR completion flag.
    CALL HIST(ITIMES,40,40,IFLAG1,IFULL)    ;Request 40 values in
                                            ; ITIMES.
    CALL SETR(5,3,0.,IFLAG2)                ;Start clock at 100 Hz in
                                            ;external event from base
                                            ;0 mode.
    CALL LWAIT(IFLAG1,0)                    ;Wait until all the samples
                                            ;have been collected.
    CALL SETR(-1,,,)                        ;Stop clock when done.
        .                                   ;Continue the program to
        .                                   ;process or store the data.
        .
```

IADC

1.4.10 IADC (Single Analog-to-Digital Conversion) Routine

IADC acquires a single digitized value from an A/D converter. (You can acquire multiple digitized values using the RTS routine.) The program requests conversion, pauses within the IADC routine until the conversion is complete, and returns the digitized value as an unsigned integer (0 to 1023 from the AR11, or 0 to 4095 from the AD11-K or LPS11).

If the A/D converter is operating in unipolar mode, a value of zero represents 0 volts, and a value of 4095 (or 1023 for the AR11) represents the full scale voltage. If the A/D converter is operating in bipolar mode, a value of zero represents the maximum negative voltage, and a value of 4095 (or 1023) represents the maximum positive voltage. Zero voltage corresponds to a value of 2048 (or 512).

IADC is called either as a subroutine or as a function.

As a subroutine:

    CALL IADC(ichannel,[igain],ivalue#)

As a function:

    ivalue = IADC(ichannel[,igain])

The arguments are:

ichannel   The logical channel number of the A/D converter.

                The channel numbers are:

Value	Meaning
0 through 63	Channel numbers for the LPS11 with LPS11-E or AD11-K with AM11-K used in single-ended or pseudodifferential mode.
0 through 15	Channel numbers for the LPS11 or AD11-K with AM11-K with switched gain operation.
	Channel numbers for the AR11 and AD11-K.
0,2,4,,,62	Even-numbered channels for the AD11-K with AM11-K used in true differential mode.
0,2,4,,,14	Even-numbered channels for the AD11-K with AM11-K used in differential mode with switched gain operation.

igain   The gain setting. The default value for igain is
             1. The gain used for converting the value is
             returned in bits 13 and 12 of ivalue. The
             possible values of bits 13 and 12 are 00, 01, 10,
             and 11 binary (0, 1, 2, and 3 decimal). These
             correspond to gain codes of 1, 2, 3, and 4
             respectively, or to the actual gains of 1, 4, 16,
             and 64. When you request a gain of 1, bits 13 and
             12 always return as 0. For all other gains, you
             must use CVSWG to separate the converted value
             from the gain bits.

             The gain codes are:

             Value   Meaning

             0   Request autogain. The software
                         determines the optimal gain to use for
                         the A/D conversion. Autogain is
                         available only with switchable gain
                         systems (AD11-K with AM11-K, or LPS11
                         with LPSSG).

             1   Request gain of 1. This value specifies
                         bipolar mode for the AR11.

             2   Request gain of 4 to multiply the input
                         signal by 4. This value specifies
                         bipolar mode for the AR11.

             3   Request gain of 16 to multiply the input
                         signal by 16. This value specifies
                         unipolar mode for the AR11.

             4   Request gain of 64 to multiply the input
                         signal by 64. This value specifies
                         unipolar mode for the AR11.

ivalue   The name of a variable containing the digitized
             value in bits 0 through 11 (or 0 through 9 for the
             AR11) and the gain code in bits 13 and 12. If you
             use IADC as a subroutine, you must include the
             argument ivalue to recover the data. If you use
             IADC as a function, the argument ivalue is
             redundant (because ivalue is returned as the value
             of the function) and can be omitted.

Error conditions:

    SYNTAX ERROR
        There are no arguments, or there are more than three
        arguments.

    ARGUMENT ERROR
        The channel number is greater than 15 for the AR11 or AD11-K
        without AM11-K.
        The channel number is greater than 63 for the AD11-K with
        AM11-K, or the LPS11.
        The gain setting is negative or greater than 4.

DEVICE CONFLICT
Autogain was requested for the AR11 by specifying igain as zero.

ADC CONFLICT
A/D conversion is already being done under control of RTS.

Interactions:

You cannot write a program that would attempt to call IADC and RTS at the same time. IADC and RTS use the same conversion hardware, and only one of the routines can execute at one time.

When the value of igain is not 1, IADC returns a value that is a combination of the digitized value and the gain setting. You can unpack the digitized value in real-number form and the gain used by calling the function CVSWG (see Section 1.4.2).

With gains other than 1, the A/D conversion system multiplies the input signal by 4, 16, or 64. When you get the digitized value from CVSWG, you must divide by the appropriate gain (4, 16, or 64) if you need to restore the value to the original scale.

Restrictions:

Autogain mode does not work with the AD11-K or the LPS11 in unipolar mode.

Autogain is not available for the AR11.

When you specify any gain other than 1, the channel number requested must be in the range 0 to 15.

The AD11-K with AM11-K hardware can be set up with gains of 1, 5, and 50 instead of 1, 4, 16, and 64. Autogain does not work in this case. The meaning of gain codes 2, 3, and 4 depends on the actual gain selected for each group of channels in the AM11-K.

Examples:

1. Collect a digitized reading from channel 2 with a gain of 1.

    VALUE=CVSWG(IADC(2))

2. Collect a digitized value from channel 1 with autogain and convert the value to the original scale.

    VAL=CVSWG(IADC(1,0),IGAIN)
    VAL=VAL/(2**(2*(IGAIN-1)))

IDIR

1.4.11 IDIR (Digital Input Reading) Routine

IDIR provides four distinct capabilities, two of which are related. The description of IDIR is presented in three sections, one for each of the classes of capabilities. The first section describes using IDIR to read a digital input register or a memory location. The second describes using IDIR for bit and byte manipulation. The third describes using IDIR to read the software clock.

Reading a Digital Input Register or a Memory Location

IDIR reads one data word from a digital input register (DR11-K or LPSDR-A) or from a memory location. It performs a logical AND using the data value and a mask specified in the argument list.

IDIR is called as a function.

ivalue = IDIR([isource],iunit,imask,[itype][,iwhere#])

The arguments are:

ivalue   The masked register or memory value returned by IDIR. IDIR performs a logical AND using imask and the register or memory value, and returns the result as the value of the function.

isource   The code specifying whether to read a register or a memory value. The default value of isource is zero.

The location codes are:

Value	Meaning
zero	Read the register specified by the logical unit number in iunit. In the process of reading the register, IDIR clears those bits (and only those bits) that it finds set.
nonzero	Read the memory location whose address appears in iunit.

iunit   The register number or the address of the value to be read.

The possible values for iunit are:

Value	Meaning
0 through 7 or 0 through 8	Logical unit numbers for the digital input registers. The 0-through-7 values select one of up to eight DR11-K digital I/O interfaces. If the system

Value	Meaning
	configuration includes an LPS11 with LPSDR-A, then 0 selects the LPSDR-A and 1 through 8 select one of up to eight DR11-K digital I/O interfaces. (The isource argument must be zero.)
0,2,...,"177776	Memory address selected. The address must be an even number. (The isource argument must be nonzero.) You can read the control/status register (CSR) or some other register associated with a device. For example, you can read the AA11-K or LPSVC status bits, or you can read registers for devices that are not supported by the FORTRAN extensions.

imask   The value used to mask the contents of the register or memory location. A commonly used mask is one with all bits on. You can request all bits on either as -1 (decimal) or as "177777.

itype   The data format of the register or memory location. The default value of itype is zero.

The data types are:

Value	Meaning
zero	Read unsigned BCD data from the register or memory location and convert it to binary. (The data word is masked before it is converted to binary.)
nonzero	Read binary data from the register or memory location.

iwhere   Bit position (0 to 15) of the rightmost nonzero bit in the portion of the register or memory location that was masked with zeros. That is, IDIR performs a logical AND using the register value and the complement of the mask, and scans the result from right to left for a nonzero bit. If no bits were set in the part masked with zeros, then iwhere contains the value -1.

Error conditions:

SYNTAX ERROR
  There are fewer than four or more than five arguments specified.

ARGUMENT ERROR
>   The memory address specified in iunit is an odd number.
>   The register requested in iunit does not exist.

Interaction:

>   None

Restrictions:

>   None

Examples:

1. Reading a digital input register.

>   IREG = IDIR(0,0,"1777,1,ISPOT)

If the register contains the octal value 51324

>   0 101 001 011 010 100

and the mask is 1777 (octal)

>   0 000 001 111 111 111

then the logical AND using the register and the mask is octal 1324.
Because binary conversion was specified by imode = 1, IDIR returns
1324 as its value.

>   0 000 001 011 010 100

IDIR also examines the logical AND using the register and the
complement of the mask:

>   .not. mask   1 111 110 000 000 000
>   register   0 101 001 011 010 100
>   result   0 101 000 000 000 000

Scanning the result from right to left, IDIR finds the   first   nonzero
bit in bit 12 and returns 12 as the value of ISPOT.   (Remember that
the rightmost bit in the word is bit 0.)

2. Reading a memory location.
You want to obtain the contents of the status register for DR11-K unit
zero.

>   MEMLOC=IDIR(-1,"167770,"177777,1)

Bit and Byte Manipulation of a Single Word

IDIR provides limited bit manipulation capability in FORTRAN.   You can
swap the bytes of a single word or shift bits left or right within a
word.

IDIR is called as a function.

>   ivalue = IDIR(,iflag,ibefore,iop[,iresult#])

The arguments are:

>   ivalue   The byte-swapped value or the logical value of the
>   last bit shifted, depending on the value of   iop.

iflag   The flag specifying the bit-manipulation and clock functions of IDIR. The value of iflag must be negative.

ibefore   The value on which bit or byte manipulation is performed.

iop   The manipulation operation selected.

The manipulation operations are:

Value	Meaning
0	Swaps the upper and lower bytes of ibefore, and returns the new value as the value of IDIR. The byte-swapped value is also returned as the value of iresult if that argument was specified. The value of ibefore does not change.
m = 1 to 16	Shifts ibefore m places to the right and returns the shifted value in iresult. IDIR fills iresult from the left with m zeros. The bits shifted out are lost. The value in ibefore does not change.
	The logical value of the last bit shifted out is returned in ivalue (the value of the function IDIR). The value returned is -1 or "true" if the bit was on, and 0 or "false" if the bit was off.
m = -1 to -16	Shifts ibefore m places to the left and returns the shifted value in iresult. IDIR fills iresult from the left with -m zeros. The bits shifted out are lost. The value in ibefore does not change.
	The logical value of the last bit shifted out is returned in ivalue (the value of the function IDIR). The value returned is -1 or "true" if the bit was on, and 0 or "false" if the bit was off.

iresult   The value resulting from byte-swapping or bit-shifting ibefore.

Error conditions:

SYNTAX ERROR
     There are fewer than four or more than five arguments present.

Interaction:

     None

Restrictions:

     None

Example:

1. You want to find out whether or not bit 13 of digital input
register 2 has been set. You have previously read the whole register
value in binary with the following call:

        IREG=IDIR(,2,-1,1)

You test bit 13 by shifting the register value three places to the
left so that bit 13 is the last bit shifted out. You are not
interested in the shifted register value itself, so you omit the last
argument, iresult:

        IBIT=IDIR(,-1,IREG,-3)

Reading the Software Clock

IDIR reads the current value of the software clock and performs a
logical AND using the clock value and a mask provided in the argument
list. (See Section 1.2.2 on the software clock.)

IDIR is called as a function.

        iclock = IDIR(,iflag,imask,)

                or

        ivalue = IDIR(,iflag,imask,,iclock#)

The arguments are:

> ivalue   The masked clock value returned as the value of
>   IDIR. The same value is returned in iclock if
>   that argument is included in the call. IDIR
>   performs a logical AND using the software-clock
>   value and imask. The value of the software clock
>   does not change.
>
> iflag   The flag specifying the clock and bit functions of
>   IDIR. The value of iflag must be negative.
>
> imask   The value used to mask the software-clock value.
>   IDIR performs a logical AND using the
>   software-clock value and imask. Request a mask
>   with all bits on either as -1 (decimal) or as
>   "177777.
>
> iclock   The masked clock value. (This value is also
>   returned as the value of the function.)

Error conditions:

> SYNTAX ERROR
>   There are fewer than four or more than five arguments
>   present.

Interaction:

> SETR is responsible for the software clock. The software clock
> is running only if you have called SETR to turn on the
> programmable clock counter in repeated interval mode.

Restrictions:

IDIR and IDOR can both read the software clock. This redundant capability is necessary because there is a restriction on subroutines called from completion routines. A FORTRAN program and a completion routine designated in the program cannot both call the same subroutine. The redundancy of IDIR and IDOR allows you to read the software clock both from within the program and from within a completion routine. For example, you could use IDIR to read the clock within the program, and IDOR to read and maybe reset the clock within a completion routine.

Examples:

1. Reading the software clock

```
    .
    .
    .
    IFL=0                   ;End flag for SETR.
    CALL SETR(4,1,1.,IFL)   ;Run the clock in repeated interval
                            ;mode at 1 msec rate.
    .
    .
    .
    ICLOCK=IDIR(,-1,-1,)    ;Read 16 bits of software clock.
```

IDOR

1.4.12 IDOR (Digital Output Register) Routine

IDOR provides three separate output capabilities, two of which are related. Each of these output capabilities is analogous to an input capability of the function IDIR. The description of IDOR is presented in two sections. The first section describes using IDOR to load a digital output register or memory location. The second section describes using IDOR to read and reset the software clock.

Loading a Digital Output Register or Memory Location

IDOR changes the value of a digital output register or memory location. The new value of the register or location is determined by the value of the arguments iselect and iset, as well as by the original value of the register or location. The following algorithm describes the function of the IDOR routine.

The bits set in iselect specify the bits in the register or location to be altered. If the bit in iselect is 0, the corresponding bit position in the register is not changed; if the bit in iselect is 1, the corresponding bit position in the register is altered according to the value at that bit position in iset.

The values of the bits in iset specify the action taken for the bits specified in iselect. If the selected bit in iset is 0, the corresponding register bit is cleared; if the selected bit in iset is 1, the corresponding register bit is set.

If all bits in iselect are 1, then all bits in the register are altered according to the values of the corresponding bits in iset. If all bits in iset are 0, then all bits in the register corresponding to those set in iselect are cleared. For example, if iselect is -1 (all bits are on) and iset is zero (all bits are off), then the entire register is cleared.

The logical relationship among the arguments is shown in the following equation:

   ivalue = (iset.AND.iselect).OR.(.NOT.iselect.AND.(old register))

IDOR is called as a function or as a subroutine.

As a function:

   ivalue = IDOR([idest],iunit,iselect,iset[,ioutput#])

As a subroutine:

   CALL IDOR([idest],iunit,iselect,iset[,ioutput#])

The arguments are:

      ivalue   The value returned by the function IDOR. The value of the function is the new value loaded into the digital output register or memory location. The new value is a combination of iselect, iset, and the original contents of the register or memory location.

idest   The destination (digital output register or memory location) of the output value. The default value of idest is zero.

Value	Meaning
zero	Load the register specified by the logical unit number in iunit.
nonzero	Write to the memory location whose address appears in iunit.

iunit   The logical unit number or the address of the location to be loaded.

Value	Meaning
0 through 7 or 0 through 8	Logical unit numbers for the digital output registers. The 0-through-7 values select one of up to eight DR11-K digital I/O interfaces. If the system configuration includes an LPS11 with LPSDR-A, then 0 selects the LPSDR-A, and 1 through 8 select one of up to eight DR11-K digital I/O interfaces. (The idest value must be zero.)
0,2,...,"177776	Memory address selected. The address must be an even number. (The idest value must be nonzero.)

iselect   The bits to be altered in the output register or memory location. The bits that are on select the corresponding bits in the register or memory location to be altered. To select all bits, you can use either -1 (decimal) or "177777.

iset   The action taken for each bit selected by iselect for alteration. If the bit in iset is 0, the selected bit is cleared; if the bit in iset is 1, the selected bit is set.

ioutput   The value returned by the function IDOR. The ioutput argument contains the value loaded into the digital output register or memory location. The following relationship defines the value of ioutput:

ioutput = (iset.AND.iselect).OR.(.NOT.iselect
             .AND,(old register))

Error conditions:

SYNTAX ERROR
     There are fewer than four or more than five arguments present.

ARGUMENT ERROR
    The DR unit addressed does not exist.
    The memory address specified was not an even number.

Interaction:

    None

Restrictions:

    None

Examples:

1. Clearing a register or memory location.

(a) Clearing a register. The call

    CALL IDOR(,1,-1,0)

clears logical register 1. Notice that the ioutput argument is omitted. Omit the ioutput argument when you have no need for the value that was loaded into the register.

(b) Clearing a memory location. The call

    IMEM=IDOR(15,"177xxx,-1,0)

clears memory location 177xxx. The variable IMEM contains the value zero as a result of the call.

2. Loading a digital output register.

You want to set the even-numbered bits in the low-order byte of the register, and clear the even-numbered bits in the high-order byte of the register. The odd-numbered bits in the register remain unchanged. Suppose the current value of the register is 177400.

    CALL IDOR(,0,"52525,"377,INEWR)

Notice that idest is defaulted (indicating a register function) and iunit is zero (indicating logical register 0). The value of iselect is 52525 and iset is 377 (both octal).

The following shows the binary values of the arguments and the action of the call on the register value:

	Binary Value
iselect	0 101 010 101 010 101
iset	0 000 000 011 111 111
old register	1 111 111 100 000 000
new register	1 010 101 001 010 101

The new value loaded into the register is 125125 (octal) and is returned as the value of INEWR.

Reading and Resetting the Software Clock

IDOR reads the current value of the software clock and performs a logical AND using the clock value and a mask provided in the argument list (see Section 1.2.2 on the software clock). IDOR can change the software clock value to a value provided in the argument list.

IDOR is called as a function or as a subroutine.

As a function:

    ivalue = IDOR(,iunit,imask,[iset][,iclock#])

As a subroutine:

    CALL IDOR(,iunit,imask,[iset],iclock#)

The arguments are:

    ivalue   The value returned by the function IDOR. IDOR
                performs a logical AND between imask and the
                current value of the software clock. After
                reading the clock value, IDOR loads the software
                clock with the value in iset, if the iset argument
                is specified.

    iunit   The flag selecting the clock function of IDOR.
                The value of iunit must be negative.

    imask   The value used to mask the current value of the
                software clock. IDOR performs a logical AND
                between imask and the clock value. To request a
                mask with all bits on, you can use either -1
                (decimal) or "177777.

    iset   The value loaded into the software clock by IDOR.
                IDOR sets the clock with the value iset. If you
                omit iset from the argument list, IDOR does not
                change the clock value.

    iclock   The masked clock value. IDOR performs a logical
                AND between imask and the current clock value.

Error conditions:

    SYNTAX ERROR
        There are fewer than four or more than five arguments
        present.

Interaction:

    None

Restrictions:

    IDOR and IDIR can both read the software clock. This redundant
    function is necessary because there is a restriction on
    subroutines called from completion routines. A FORTRAN program
    and a completion routine designated in the program cannot both
    call the same subroutine. The redundancy of IDOR and IDIR allows
    you to read the software clock both from within the program and
    from within a completion routine. For example, you could use
    IDOR to read and reset the clock within a completion routine, and
    IDIR to read the clock within the program.

Examples:

1. You want to read the 16-bit software clock without changing its
value.

        ICLOCK=IDOR(,-1,-1,)

2. You want to read the rightmost eight bits of the software clock,
and reset the clock to zero.

        ICLOCK=IDOR(,-1,"377,0)

KB2BCD

1.4.13 KB2BCD (Convert Binary to BCD) Routine

KB2BCD converts a 16-bit binary integer in the range 0 to 9999 (decimal) to 4-digit unsigned BCD form.

KB2BCD is called as a function.

    idigits=KB2BCD(ibinary)

The arguments are:

idigits   The 4-digit unsigned BCD representation of ibinary.

ibinary   The standard binary integer value to be converted. The value of ibinary must be in the range 0 to 9999.

Error conditions:

SYNTAX ERROR
    There is not exactly one argument present.

ARGUMENT ERROR
    The value to be converted (ibinary) is negative or greater than 9999.

Interaction:

You can use IDOR to output unsigned BCD values returned by KB2BCD.

Restrictions:

None

Example:

1. The decimal value of IBIN is 4175 (or 10117 octal):

    0 001 000 001 001 111

You want to convert IBIN to BCD form for output to a BCD instrument.

    IBCD=KB2BCD(IBIN)

IBCD contains the BCD value for 4175 (shown with BCD grouping of the bits):

    0100 0001 0111 0101

KBCD2B

1.4.14 KBCD2B (Convert BCD to Binary) Routine

KBCD2B converts a 4-digit unsigned BCD input to 16-bit binary integer
form. The result is always positive.

KBCD2B is called as a function.

    ibinary=KBCD2B(idigits)

The arguments are:

    ibinary   The 16-bit binary integer form of idigits.

    idigits   The 4-digit (16-bit) unsigned BCD input value.

Error conditions:

    SYNTAX ERROR
        There is not exactly one argument present.

Interaction:

    None

Restrictions:

    None

Example:

1. Suppose IBCD contains the following bit pattern, which is 0969 in
BCD:

    0000 1001 0110 1001

As a result of the following call,

    IBIN=KBCD2B(IBCD)

IBIN contains the following bit pattern, which is 969 decimal.

    0 000 001 111 001 001

LED

1.4.15 LED (LED Display) Routine

LED displays numbers on the LED panel of the LPS11.   You cannot use
LED without an LPS11 in the system configuration.   There are six LED
display positions and a movable decimal point.

LED is called as a subroutine.

        CALL LED(ivalue,'format')

                 or

        CALL LED(rvalue,'format')

The arguments are:

    ivalue   The integer value to be displayed in the format
                    specified.

    rvalue   The real value to be displayed in the format
                    specified.

    format   The FORTRAN format specification for the value to
                    be displayed. The maximum field width for integer
                    (I) formats is six; the maximum field width for
                    real (F) formats is seven. The maximum F format
                    field width is wider to allow for a decimal point.
                    The decimal point does not use an LED display
                    position as the decimal point appears between
                    display positions. The minus sign does use an LED
                    display position so that the largest negative
                    numbers that can be displayed have five
                    significant digits. The largest positive numbers
                    that can be displayed have six significant digits.
                    If the value exceeds the width of the field
                    specified, the display contains all minus signs.

Error conditions:

    SYNTAX ERROR
        There are not exactly two arguments present.

    ARGUMENT ERROR
        The format specified is not legal.

    DEVICE CONFLICT
        The system configuration does not include an LPS11.

Interaction:

        LED can be used to display data values acquired by other routines
        (for example, IADC, IDIR).   You might want to do this to
        calibrate your instruments or Schmitt-trigger thresholds prior to
        a data collection run.

Restrictions:

        LED requires an LPS11.

Examples:

1. Display a negative integer.

```
NEG=-123
CALL LED(NEG,'I4')
```

CALL LED(NEG,'I3')

2. Display a real number.

```
REALV=25.94
CALL LED(REALV,'F5.2')
```

3. Display a single analog value from channel 0 of the ADC. Assume that you want the display value to be expressed in volts and that the ADC accepts bipolar input with a range of -5 volts to +5 volts. The following example shows the transformation from the 12-bit value acquired to the corresponding input voltage.

```
CALL LED((FLOAT(IADC(0)-2048)/2048.)*5.,'F7.4')
```

LWAIT

1.4.16 LWAIT (Wait) Routine

LWAIT initiates a pause in the program. Any activity already started by the program (for example, repeated sampling) continues, but execution of the program does not continue past the LWAIT statement until the arguments differ. For example, use LWAIT to compare a numeric value to a completion flag (that will be changed by an interrupt service or completion routine).

LWAIT is called either as a subroutine or as a function.

As a subroutine:

        CALL LWAIT(ivaluel,ivalue2)

As a function:

        ivar=LWAIT(ivaluel,ivalue2)

The arguments are:

> ivaluel
>
> > The name of a variable to be compared to ivalue2. If you use LWAIT as a function call, LWAIT sets ivar to ivaluel.
>
> ivalue2
>
> > A variable or constant to be compared to ivaluel.

> NOTE
>
> > Neither of the arguments to LWAIT is changed within LWAIT, but at least one of the arguments specified must be a variable that changes as a result of activity in an interrupt service or completion routine.

Error conditions:

> None

Interaction:

> At least one of the arguments for LWAIT must be changed by an interrupt service or completion routine. For example, an argument for LWAIT can be a completion flag for one of the routines SETR, RTS, HIST, or DRS.

Restrictions:

> One of the arguments must eventually change if program execution is to continue. The call
>
>   CALL LWAIT(1,1)
>
> would suspend program execution indefinitely.

Example:

1. You want a timed program delay of 500 msec.

```
        .
        .
        .
    IFLAG=0                         ;Initialize flag.
    CALL SETR(4,0,500.,IFLAG)       ;Start clock in single interval
                                    ;mode for 500 msec.
    CALL LWAIT(IFLAG,0)             ;Wait until clock endflag is
                                    ;nonzero.
```

1.4.17 REL (Relay Control) Routine

REL operates the two normally open relays in the LPS11. (See the LPS11 hardware manual for considerations on connecting equipment to the relays.)

REL is called as a subroutine.

    CALL REL(irelay,isetting)

The arguments are:

irelay
: The relay number. The relays are numbered 1 and 2.   If irelay is an odd number, REL selects relay 1; if irelay is an even number, REL selects relay 2.

isetting
: The relay setting.

Value	Meaning
zero	Open relay.
nonzero	Close relay.

Error conditions:

SYNTAX ERROR
    There are not exactly two arguments present.

ARGUMENT ERROR
    The relay number is negative.

DEVICE CONFLICT
    The system configuration does not include an LPS11.

Interaction:

None

Restrictions:

REL is meaningful only if the system includes an LPS11.

Examples:

1.   The following call closes relay number 2:

    CALL REL(2,-5)

2.   Read digital input register unit 1. Leave relay 1 open if the register value is zero;   close relay 1 if the register value is nonzero.

    CALL REL(1,IDIR(,1,-1,1))

RTS

1.4.18 RTS (Repeated Sampling) Routine

RTS mediates repeated sampling from A/D converters. The RTS routine extends your analog sampling capabilities beyond the single reading that can be acquired using IADC.

RTS is a flexible routine providing three basic capabilities.

1. Single-sweep sampling. RTS provides the capability for acquiring limited bursts of analog data. You so specify the parameters that RTS stops sampling after it fills an array with data. For the AD11-K and the AR11, RTS provides a fast, restricted capability, single-sweep sampling mode called fast-sweep mode (described later).

2. Continuous sampling. RTS provides the capability for acquiring continuous streams of analog data. There are two classes of continuous sampling: finite continuous sampling, in which you specify the total number of samples to be acquired, and infinite continuous sampling, in which you do not specify the total number of samples in the RTS call. If you request infinite continuous sampling, you must call RTS a second time to stop the sampling process.

    In both cases, you specify the parameters so that RTS fills data arrays continuously. You either designate a completion routine that executes periodically or arrange your program code to empty the arrays. You structure the completion routine or program to dispose of the data as you require (for example, by writing it to secondary storage, or by reducing it to means or counts). Because completion routines require time to execute, the maximum continuous sampling rate (which involves a completion routine) is lower than the maximum single-sweep sampling rate (which does not involve a completion routine).

3. End sampling. You can call RTS in parameter-adjustment mode to stop infinite continuous sampling.

RTS is called with a subroutine call. The number of arguments necessary depends on the capability you are requesting.

Single-sweep sampling:

CALL RTS(iarrayname,iarraysize,,[nsamples],[ifirst],[nchannels],
        [igain],[imode],iendflag#,nleft)

Continuous sampling:

CALL RTS(iarrayname,iarraysize,[nsubarrays],[nsamples],[ifirst],
        [nchannels],[igain],[imode],iendflag#,nleft#
        [,[interval],complete])

End sampling:

CALL RTS(,iarraysize,,,,,,imode*,iendflag,nleft)

The arguments are:

iarrayname The name of the integer array in which RTS places the analog data sampled. With continuous sampling, RTS treats the data array as a circular array. If you omit iarrayname, the RTS call requests parameter-adjustment mode.

iarraysize The length of iarrayname (in words). This parameter specifies the amount of memory reserved for RTS to store data. For single-sweep sampling, the value of iarraysize should be calculated according to the following relationships:

$$iarraysize = nsamples*nchannels$$

or

$$iarraysize = nsamples*nchannels*2 \quad (for \; LPS11 \; dual \; sample-and-hold)$$

For continuous sampling, iarraysize should be greater than interval*nchannels*nsubarrays.

nsubarrays The number of logical partitions in iarrayname. The length of each partition is equal to the largest integer in (iarraysize/nsubarrays). For example, with iarraysize = 1024 and nsubarrays = 10, there are 10 partitions, each 100 words long, and 24 unused words at the end of iarrayname. The default number of partitions is one.

The number of partitions must be less than or equal to iarraysize. The length of each partition must be an integer multiple of the number of channels to be sampled.

nsamples The number of samples per channel. The parameter specifies the total number of times that each analog channel is sampled. The default value of nsamples is iarraysize. The default can be used only with single-sweep sampling for a single-channel sample. The default cannot be used with the LPS11 dual sample-and-hold.

In single-sweep sampling, the value you assign to nsamples depends on whether you are collecting data from each channel independently, or from pairs of channels with LPS11 dual sample-and-hold. For normal sampling, you specify

$$nsamples = iarraysize/nchannels$$

With LPS11 dual sample-and-hold, you specify

$$nsamples = iarraysize/(nchannels*2)$$

The maximum value of nsamples for DMA sampling with the LPS11 is 4096.

In continuous sampling, the value you assign to nsamples depends on how you want to stop the sampling process.

(a) For finite continuous sampling, you set nsamples equal to the number of samples you want. In this case, the value of nsamples is greater than iarraysize/nchannels (because you are requesting continuous sampling).

(b) For infinite continuous sampling, you set nsamples equal to any negative number:

    nsamples < 0

RTS then samples data until the program executes a call to RTS in parameter-adjustment mode with a negative value for imode.

For either type of continuous sampling, you must write the program code and specify the parameters necessary to remove data from full array partitions. Continuous sampling is not valid for LPS11 DMA sampling.

ifirst   The first channel to be sampled. If you request more than one channel, the channels are sampled consecutively in ascending order so that ifirst is the lowest numbered channel to be sampled. The default value for ifirst is zero.

The valid channel numbers are:

Value	Meaning
0 through 63	Range of channel numbers for the LPS11 or AD11-K with AM11-K, without switchable gain.
0 through 15	Range of channel numbers for the LPS11 or AD11-K with AM11-K, with switchable gain. Range of channel numbers for the AR11 or AD11-K.
0,2,...,62	Range of channel numbers for AD11-K with AM11-K with differential input, without switchable gain.
0,2,...,14	Range of channel numbers for AD11-K with AM11-K used in differential mode with switchable gain.
0,2,...,7	Range of channel numbers for LPS11 with dual sample-and-hold.

nchannels   The number of consecutive channels to be sampled in either single-ended or differential mode. The value of nchannels must be the total number of channels between the lowest numbered channel to be sampled and the highest numbered channel to be sampled (inclusive). That is,

    nchannels = (highest numbered - lowest
                 numbered) + 1

This is the case, regardless of whether the data from any of the intervening channels are of interest.

For the LPS11 dual sample-and-hold, nchannels is the number of consecutive pairs of channels to be sampled.

The default value for nchannels is 1.

igain   The gain setting. The default value for igain is 1. The gain used for converting each value is returned in bits 13 and 12 of ivalue. The possible values of bits 13 and 12 are 00, 01, 10, and 11 binary (0, 1, 2, or 3 decimal). These correspond to gain codes of 1, 2, 3, and 4 respectively, or to the actual gains of 1, 4, 16, and 64. When you request a gain of 1, bits 13 and 12 always return as 0. For all other gains, you must use CVSWG to separate the converted value from the gain bits.

Value	Meaning
0	Request autogain. The software determines the optimal gain to use for each A/D conversion. Autogain is available only with switchable gain systems (AD11-K with AM11-K, or LPS11 with LPSSG).
1	Request gain of 1. This value specifies bipolar mode for the AR11.
2	Request gain of 4 to multiply the input signal by 4. This value specifies bipolar mode for the AR11.
3	Request gain of 16 to multiply the input signal by 16. This value specifies unipolar mode for the AR11.
4	Request gain of 64 to multiply the input signal by 64. This value specifies unipolar mode for the AR11.

imode   The mode of sampling.

The stop codes that terminate sampling and determine what happens to completion routines when sampling stops.

The default for imode is zero.

The mode values and their meanings are:

Value	Meaning
0	For the AD11-K and the LPS11, take one A/D sample on each Schmitt trigger #1 interrupt. For the AR11, take one A/D sample on each external event.
2	Take one A/D sample on each clock-overflow pulse.
4	For the AD11-K and the AR11, enable fast-sweep mode (see the first restriction in this section). Take one A/D sample on each Schmitt trigger #1 interrupt (for the AD11-K) or on each external event (for the AR11). For the LPS11, enable dual sample-and-hold. Take one simultaneous pair of A/D samples on each Schmitt trigger #1 interrupt.
6	For the AD11-K and the AR11, enable fast-sweep mode (see the first restriction in this section). Take one A/D sample on each clock overflow pulse. For the LPS11, enable dual sample-and-hold. Take one pair of A/D samples on each clock overflow pulse.
8	Enable DMA. DMA is valid only for the LPS11 with single-channel single-sweep sampling.
9	Enable DMA in burst mode. In burst mode, data are acquired as fast as they can be converted. That is, one conversion starts as soon as the previous one is completed. Burst mode is valid only for the LPS11 with single-channel single-sweep sampling.

For the LPS11, dual sample-and-hold, DMA, and clock-overflow sampling can be requested singly (by using the code in the table) or in combination by adding the codes for the desired options to form the value of imode. For example, to initiate clock-overflow (mode 2), dual sample-and-hold (mode 4) sampling for the LPS11, specify imode = 2 + 4 = 6.

The stop codes and their meanings are:

Value	Meaning
-4	Continue sampling only to fill the current array partition. Do not queue any more completion-routine requests, but allow those requests already queued to run.
-3	Continue sampling only to fill the current array partition. Do not honor any queued completion-routine requests.
-2	Stop sampling initiated by RTS. Do not queue any more completion-routine requests, but allow those requests already queued to run.
-1	Stop sampling initiated by RTS. Do not honor any queued completion-routine requests.

iendflag   The completion and/or error flag. You must set iendflag to zero before you call RTS. RTS increments iendflag either when all samples requested have been acquired or when sampling has been stopped by a stop code (negative value for imode). If a data overrun or other error occurs, RTS sets iendflag to a negative value.

nleft   The array partition flag. RTS initializes nleft with the number of array partitions. RTS decrements nleft by 1 each time an array partition becomes full. Thus nleft contains the number of partitions currently available to RTS. If you are using a completion routine to remove data from full array partitions, you should increment nleft each time a partition becomes available. If nleft reaches zero before the number of samples requested in nsamples has been acquired, a data overrun has occurred and RTS then sets iendflag to indicate an error.
Note: nleft is meaningful only with continuous sampling.

interval   The number of interrupts between calls to the completion routine. You specify the number of interrupts, external events, or clock overflow pulses that should occur before the completion routine is executed. RTS initializes an internal variable to interval and decrements it after each interrupt. When the value reaches zero, RTS reinitializes the internal variable, and the completion routine executes. Therefore, the completion routine executes every interval interrupts or events.

The default value for interval is iarraysize/nsubarrays. If only one data point is collected on each interrupt, the default ensures that the completion routine executes when each array partition becomes full. With dual sample-and-hold or more than one channel, you must calculate interval if you want the completion routine called as each array partition becomes full.

interval = iarraysize/(nsubarrays*nchannels)

or

interval = iarraysize/(nsubarrays*nchannels*2)

for LPS11 dual sample-and-hold.

complete   Completion-routine name. You write a completion routine that executes whenever interval interrupts have occurred. There is no default completion-routine name. When you omit the last two arguments, RTS does not designate a completion routine.

Error conditions:

SYNTAX ERROR
    There are fewer than 10 or more than 12 arguments present.

ARGUMENT ERROR
    The starting channel number (ifirst) is too large.
    The number of channels requested (nchannels) is negative, or too large (greater than 16 or greater than 64, depending on the hardware).
    Autogain was requested for the AR11.
    The gain code (igain) is greater than 4.
    The mode (imode) requested is out of range (less than -4 or greater than 15).
    The sum of ifirst and nchannels requests sampling from a nonexistent channel with channel number greater than 63 (or 15).
    The size of the array partition is not an even multiple of the number of channels to be sampled (nchannels).
    Autogain was requested with DMA.
    Continuous sampling with stop code termination was specified for DMA.
    Data acquired with DMA exceeds the available space (iarraysize).

DEVICE CONFLICT
    An RTS call attempted to initiate sampling while sampling from a previous call to RTS was still in progress.

Interactions:

    RTS can request an A/D conversion on each clock-overflow pulse. It does not require clock interrupts. You can run the clock without interrupts by requesting noninterrupt mode in SETR. RTS runs faster without clock interrupts because you save the time normally used to process the clock interrupts. RTS runs even faster when SETR noninterrupt mode is combined with the fast-sweep mode described in the first restriction in this section.

You cannot write a program that would attempt to call IADC and RTS at the same time. Both IADC and RTS use the same conversion hardware and only one of the routines can execute at a time.

When the value of igain is not 1, RTS returns a packed value containing the digitized value and the gain setting. You can unpack the digitized value in real-number form, and the gain in integer form by using CVSWG (see Section 1.4.2).

With gains other than 1, the A/D conversion system multiplies the input signal by 4, 16, or 64. When you get the digitized value in real-number form from CVSWG, you must divide it by the appropriate gain (4, 16, or 64) if you need to restore the value to the original scale.

Restrictions:

Fast-sweep mode is a mode available only with the AD11-K or AR11. With fast-sweep mode, you can acquire analog data at a rate several times faster than normally possible with RTS. There are several restrictions on using fast-sweep mode. You can request fast-sweep mode only for single-sweep sampling for a single-channel sample with the clock running in noninterrupt mode. There can be no array partitions, no completion routines, and no autogain.

Data overrun occurs when RTS fills the array partitions faster than the completion routine can empty them. This situation occurs when the clock rate is too fast for the size of the array partitions.

When the AD11-K, with or without AM11-K, is used in differential mode, the data in odd numbered channels duplicate those in even numbered channels, and can be ignored.

DMA sampling is available only in single-sweep mode for single-channel sampling. Continuous sampling is not valid for DMA sampling. Call RTS once for each DMA sweep required. The maximum number of data points that can be accepted in one DMA operation is 4K words. The maximum value for iarraysize with DMA is 4096.

The AD11-K with AM11-K hardware can be set up with gains of 1, 5, and 50 instead of 1, 4, 16, and 64. Autogain does not work in this case. The meaning of gain codes 2, 3, and 4 depends on the actual gain selected for each group of channels in the AM11-K.

Autogain does not work with an AD11-K or LPS11 in unipolar mode. Autogain is not available at all for the AR11.

Examples:

1. You want to collect a single sweep of 200 samples from four channels, starting with channel 0. Because it is single-sweep sampling, you omit nsubarrays, interval, and a completion routine name. You use the default values for ifirst (0), igain (1), imode (0). The nleft argument (IDUM) is not used by single-sweep sampling, but must be included to satisfy the syntax.

```
      DIMENSION IDATA(4,50)                  ;Set up the data array.
      IFLG=0                                 ;Initialize RTS flag.
      CALL RTS(IDATA,200,,50,,4,,,IFLG,IDUM) ;Call RTS.
      CALL LWAIT(IFLG,0)                     ;Wait until all samples
                                             ;read.
        .
        .
        .
```

(Notice that the data array could have been dimensioned as IDATA(200). FORTRAN stores the data the same way regardless of how you specify the dimensions. Using IDATA(4,50) can be more convenient for later data access, because the value of the first subscript specifies the channel number and the value of the second subscript specifies the sequence number of the sample.)

2. You want to request continuous sampling of 1300 samples using a 400-word data array with four 100-word partitions. You sample channel 1 100 times per second on clock overflow, with a gain of 3. This example processes the array partitions without using a completion routine. As soon as a partition is full, the program sums the data in that partition, and frees the partition by incrementing the partition count (NLEFT).

```
      DIMENSION IDATA(100,4),SUM(13)   ;Set up arrays.
      DO 5 I=1,13
5     SUM(I) = 0                       ;Initialize SUM array.
      N=0                              ;Index for SUM array.
      IPART=0                          ;Index for IDATA array.
      IRFLG=0                          ;RTS endflag.
      ISFLG=0                          ;SETR endflag.
      CALL RTS(IDATA,400,4,1300,1,1,3,2,IRFLG,NLEFT)
                                       ;Request continuous sampling.
      CALL SETR(5,1,1.,ISFLG)          ;Start clock at 100Hz.
10    IF (NLEFT.EQ.4) GO TO 10         ;Wait for first partition to
                                       ;fill.
      IPART=IPART+1                    ;No. of partitions (mod 4).
      N=N+1                            ;Increment index for SUM array.
      DO 20 I=1,100
20    SUM(N)=SUM(N)+CVSWG(IDATA(I,IPART))/16.
                                       ;Sum the data in the
                                       ;just-filled partition.
      IF (IPART.LT.4) GO TO 30         ;Check to see if end of IDATA.
      IPART=0                          ;Start indexing from start
                                       ;of array next time.
30    NLEFT=NLEFT+1                    ;Increment the available
                                       ;partition count.
      IF (IRFLG.GE.0) GO TO 40         ;Skip if not an error.
      STOP 'RTS ERROR'                 ;Error if IRFLG<0.
40    IF (IRFLG.EQ.0) GO TO 10         ;If still sampling, jump.
      CALL SETR(-1,,,)                 ;Finished, stop clock.
      TYPE 41,SUM                      ;Output the data.
41    FORMAT(F12.2)
      STOP 'DONE'
      END
```

(Notice that you could specify the data array as IDATA(400). However, your subscript calculation would then be more complicated. FORTRAN stores its arrays "by columns," so that it puts the first 100 samples into IDATA(1,1) through IDATA(100,1), the second 100 samples into IDATA(1,2) through IDATA(100,2), and so on. Therefore, the value of the first subscript specifies the sequence number within the partition, and the value of the second subscript specifies the partition number.)

1.4.19 SDIS (Stop Display) Routine

SDIS stops a continuous display that is being controlled by either DIS or DXY.

SDIS is called as a subroutine.

        CALL SDIS

There are no arguments.

Error conditions:

        None

Interaction:

        While the continuous-display routines (DIS, DXY) are running, the only other activities for which resources are guaranteed are interrupt service routines and completion routines. Therefore, the most practical way to stop a continuous display is by calling SDIS from within a completion routine. For example, you might use SETR to designate a completion routine that would be executed after a specific time interval, or on receipt of an external event.

Restrictions:

        None

Examples:

1. The following example is part of the example given in full in the description of DIS. The completion routine HOLDIT contains the call to SDIS. The completion routine is specified in the call to SETR.

        EXTERNAL HOLDIT   ;Specify the completion
                                                        ;routine.
           .
           .
           .
        IFLG=0   ;End flag for SETR.
        CALL SETR(5,0,1000.,IFLG,,HOLDIT)   ;Start the clock.
        CALL DIS(ISCREEN,ISPACE,NPTS,48,20)   ;Start the display.
        PAUSE
           .
           .
           .

        SUBROUTINE HOLDIT   ;Completion routine.
        CALL SDIS   ;Stop the display.
        RETURN
        END

SETR

1.4.20 SETR (Set Rate of the Programmable Clock) Routine

SETR provides several options for controlling time intervals and counting events. The hardware clock counter can be used to generate interrupts to the processor at specified intervals or in response to external events, to measure time intervals, or to count events.

The KW11-K and LPS clock counter run at any of five crystal-controlled frequencies (100 Hz to 1 MHz), at line frequency, or at an external frequency supplied through a Schmitt trigger. The AR11 clock counter runs at any of five crystal-controlled frequencies, or at a frequency supplied through an external input line.

The KW11-K and LPS clock counter operate in any of four programmable modes: single interval, repeated interval, external event timing, or external event timing from zero base. The AR11 clock counter has two programmable modes: single interval, and repeated interval. The clock modes are described later in this section.

SETR allows you to specify the clock mode, the clock rate (or type of synchronization), the length of the interval or number of counts, the completion routine to be called, and the conditions under which it is called. Call SETR to start or to stop the clock.

SETR is called as a subroutine.

    CALL SETR(irate,imode,rcount,iendflag#[,[interval],complete])

The arguments are:

    irate   The rate for the oscillator that controls the clock counter frequency.

                    The stop codes that disable the clock and determine what happens to completion routines when the clock is disabled.

                    Table of rate codes:

Value	Meaning
0	Stop the clock (without disabling it). No rate selected.
1	Set clock frequency to 1 MHz.
2	Set clock frequency to 100 KHz.
3	Set clock frequency to 10 KHz.
4	Set clock frequency to 1 KHz.
5	Set clock frequency to 100 Hz.

Value	Meaning
6	Use Schmitt trigger #1 (the external input line for the AR11) as the input for an external frequency source.
7	Set clock frequency to line frequency (50 Hz or 60 Hz). (Not valid for the AR11.)

The stop codes are:

Value	Meaning
-2	Disable the clock. Do not queue any more completion-routine requests, but allow those requests already queued to run.
-1	Disable the clock. Do not honor any queued completion-routine requests.

imode   The imode argument selects the mode of clock operation.
The external event and external start modes (modes 2 through 7, 12, and 13) are not valid for the AR11.

The codes for clock modes are:

Value	Meaning
0	Single-interval mode. The clock counter counts rcount times, generates an interrupt, and then stops. You need not issue a stop code because the clock turns off automatically. For the AR11, mode 0 can be used only with rates 1 through 5.
1	Repeated-interval mode. The clock counter counts rcount times, overflows (generating an interrupt), and then begins again to count rcount times. RTS and DRS require overflow events to synchronize sampling.
2	External-event timing mode. The initial value of the clock counter is rcount (which you must set to zero). The clock counter runs at the rate specified by irate. A pulse from Schmitt trigger #2 causes the current clock counter value to be transferred to a clock register. (You can read the register value into a data array using HIST.) The clock counter continues to count.

Value	Meaning

3   External-event timing mode from zero base (similar to mode 2). The initial value of the clock counter is rcount (which you must set to zero). The clock counter runs at the rate specified by irate. A pulse from Schmitt trigger #2 causes the current clock counter value to be transferred to a clock register. (You can read the clock register value into a data array using HIST.) The clock counter is cleared to zero and the clock continues to count.

External start modes.
When you add 4 to modes 0 through 3 above, you obtain a mode value that specifies an external start pulse for the clock. In modes 4 through 7, the clock counter starts counting on receipt of a pulse from Schmitt trigger #1, rather than starting as soon as the routine SETR is executed. External start modes are not available with the AR11.

The codes for external start clock modes are:

Value	Meaning

4   Single-interval mode with external start.

5   Repeated-interval mode with external start.

6   External-event timing mode with external start.

7   External-event timing mode from zero base with external start.

Noninterrupt modes.
When you add 8 to modes 0, 1, 4, and 5 above, you obtain a mode value that specifies running the clock counter so that clock overflow generates a clock overflow pulse but does not generate an interrupt. The RTS routine does not require the clock-overflow interrupt, but only the clock-overflow pulse. RTS runs faster with the clock interrupts turned off because you save the time normally required to process the interrupts. The clock-overflow interrupts are disabled when the clock mode is 8, 9, 12, or 13.

HIST and DRS, which require clock interrupts, do not work with noninterrupt mode. In noninterrupt mode, SETR does not run the software clock.

The codes for noninterrupt mode are:

Value	Meaning
8	Single-interval noninterrupt mode.
9	Repeated-interval noninterrupt mode.
12	Single-interval noninterrupt mode with external start.
13	Repeated-interval noninterrupt mode with external start.

Limited mode.
When you add 8 to modes 9 and 13 above, you obtain a limited service clock mode. In limited mode, the only function SETR performs is to increment the software clock. If the software clock is the only timing function you require, use limited mode.

None of the other services requiring SETR interrupts (HIST, DRS, and SETR completion routines) is available in limited mode.

The codes for limited mode are:

Value	Meaning
17	Repeated-interval limited mode.
21	Repeated-interval limited mode with external start. (External start modes are not available for the AR11.)

rcount   Initial count value. The clock counter counts rcount times at the frequency specified by irate, and then overflows and generates an interrupt. Therefore, the clock generates an interrupt every rcount counts, resulting in an interrupt rate equal to irate/rcount. The value of rcount must be a real number. For the KW11-K or the LPS11, the maximum value for rcount is 65535 (decimal); for the AR11, the maximum value for rcount is 255 (decimal). With the external event timing modes, rcount must be zero.

iendflag   The completion flag for single interval mode. SETR increments iendflag only when the clock counter overflows in mode 0. You must set iendflag to zero before you call SETR.

interval   The number of interrupts between calls to the completion routine. With this argument, you specify the number of clock overflow interrupts or the number of Schmitt trigger pulses elapsing before the completion routine executes. SETR has an internal variable initialized to interval that it decrements after each clock interrupt or Schmitt trigger #2 pulse. When the value reaches

zero, SETR calls the completion routine. The completion routine thus executes after every interval interrupts. The default value of interval is 1.

complete   Completion-routine name. The completion routine that you provide is executed whenever interval interrupts have occurred. There is no default completion-routine name. When you omit the last two arguments, SETR does not call a completion routine.

Since the completion routine is called from the clock interrupt service routine, you cannot request a completion routine when the clock is running in noninterrupt mode.

Error conditions:

SYNTAX ERROR
There are fewer than four or more than six arguments present.

ARGUMENT ERROR
The rate requested is less than -4 or greater than 7.
The mode requested is not one of the valid modes described above.
The count value requested for the AR11 is greater than 255.

DEVICE CONFLICT
Consecutive calls to SETR with rate parameter 1 through 7. This error does not occur for consecutive calls with variants of single-interval mode (modes 0, 4, and 8). The remedy is to insert (between the other calls) a call to SETR using a negative rate.

CLOCK ERROR--RATE TOO FAST
Clock overrun occurred, meaning that a clock interrupt has been lost and program execution stops. The remedy is to request a lower clock rate, or, if practical, to run in noninterrupt mode.

Interactions:

Use SETR to get the clock going in the desired mode at the desired rate. You can then use RTS, DRS, or HIST to acquire analog, digital, or time interval data (respectively).

SETR controls the operation of the 16-bit software clock (see Section 1.2.2).

DRS and HIST require clock-overflow interrupts. Therefore those routines do not operate properly if the clock is running in noninterrupt mode.

Restrictions:

You can use the following call to stop the clock (except in modes 0 and 4 in which it stops automatically). In this case, you do not need to specify names or values for the second through the fourth arguments.

        CALL SETR(-1,,,)

Examples:

1.   CALL SETR(6,0,2.0,IFLAG,,NAME)   ;Requests completion routine
                                              ;NAME after Schmitt trigger
                                              ;#1 has fired twice.

1.5 BUILDING A REAL-TIME SUPPORT LIBRARY

When your system is delivered, you receive the FORTRAN extensions real-time support routines in object module form, and a configuration routine in MACRO source form. (Appendix B lists the names and contents of these files.) Before you can use the real-time support routines, you must combine the object modules you received into a library. Before you create the library, you must be sure either that your system configuration matches that specified in the configuration routine, or that you have changed the configuration routine so that it describes your system.

1.5.1 System Configuration

The real-time support library assumes that you have a standard system unless you inform it otherwise by changing the necessary source statements in CONFIG.MAC. The source program CONFIG.MAC is a MACRO program containing a complete hardware description of a "standard" system.

1.5.1.1 A Standard System - To find out what a standard system contains, list the source file CONFIG.MAC. The first few pages of the listing contain the configuration set-up block, with Section A containing the hardware definition, and Section B containing the interrupt vector and status register addresses. Each statement defines some aspect of the system. For example,

    $AD11K=0

is the statement defining the absence of an AD11-K A/D converter and KW11-K programmable clock counter in the system.

1.5.1.2 Changing Configuration Specifications - Each statement in CONFIG.MAC is accompanied by comments that explain the statement, and the changes to make if your system differs from the standard system. Read through the explanation, and, using the text editor on your system, make whatever changes are required to CONFIG.MAC so that it describes your system. For example, if you have the AD11-K and KW11-K on your system, change the definition statement to

    $AD11K=1

Any changes you make will probably be to the hardware definition statements. The interrupt vector and status register addresses are factory-set standard addresses. Do not change any of these address statements in CONFIG.MAC unless you have made the corresponding hardware changes.

1.5.1.3 **Assembling the Revised Configuration** - The changes you make to CONFIG.MAC are changes to a source file. To incorporate the changes into the system, you must assemble the revised source file using the MACRO assembler. Your dialogue with the computer has the following sequence (where your input has been underlined):

```
.MACRO/LIST CONFIG
ERRORS DETECTED: 0
.
```

1.5.2 **The Real-Time Support Library**

Before you can use the real-time support subroutines, you must create the real-time subroutine library. Create the library file using the librarian system utility LIBR. (Refer to the RT-11 System User's Guide for more information on the librarian.)

1.5.2.1 **Generating the Library** - Use the LIBR program to generate the library of real-time subroutines.

```
.LIBR/CREATE
Library?  libnam
Files  ?  LDPOBJ
.
```

If you have the standard system configuration, you can now use the library that you created (libnam).

If you have a nonstandard system configuration and have made the necessary changes in CONFIG.MAC, you must perform one further step to generate the library. Still using the LIBR program, revise the library, libnam, to include the changes in CONFIG.

```
.LIBR
Library?  libnam
Files  ?  CONFIG/REPLACE
```

1.5.2.2 **Accessing the Library** - Write your FORTRAN programs using the real-time subroutine and function calls described in this chapter. Compile the programs in the normal manner. (Refer to the RT-11 System User's Guide for information on the FORTRAN compiler and its optional switches.)

When you link the programs, include the name of your real-time library in the list of commands for the linker:

```
.LINK progrm,libnam
```

(if you have one system library SYSLIB), or

```
.LINK progrm,libnam/F
```

(if FORLIB and SYSLIB are separate libraries).

1.6 ERROR MESSAGES

The FORTRAN extensions routines produce five error messages. These messages are SYNTAX ERROR, ARGUMENT ERROR, DEVICE CONFLICT, ADC CONFLICT, and CLOCK ERROR. Each error message appears with a line number, indicating the line where the error occurred. You must refer to your listing to determine which routine caused the error, and then refer to the error summary table (Appendix C) to determine the possible causes of the error.

The error summary table uses the following abbreviations for the error conditions:

Abbreviation	Error Condition
SYN	SYNTAX ERROR The call to the subroutine or function contains an invalid number of arguments.
ARG	ARGUMENT ERROR At least one of the arguments in the call to the routine has an illegal value. The error message does not indicate which argument is in error.
DEV	DEVICE CONFLICT This error arises when one of the routines tries to use a device that is busy. This can happen under the foreground/background monitor if another job is using the device. It can also occur when you call a routine for which completion-routine requests are still queued as a result of a previous call to the routine.
ADC	ADC CONFLICT Device conflict error for the A/D converter: the A/D converter is busy.
CLK	CLOCK ERROR--RATE TOO FAST The clock interrupts are occurring too fast to be processed.

Error-causing conditions are described in the reference section for each routine. A summary table of error conditions for all routines appears as Appendix C.

CHAPTER 2

FORTRAN DEBUGGING TECHNIQUE (FDT)

2.1 INTRODUCTION

The FORTRAN Debugging Technique (FDT) is a sophisticated interactive debugging tool for FORTRAN IV programs. FDT gives you step-by-step control of the execution of your program and the ability to examine and change the contents of any variable in your program during program execution.

FDT runs on any PDP-11 with FORTRAN IV under the RT-11 or RSTS/E operating system. FDT requires approximately 2K words of memory space during a debugging session.

To use FDT successfully, you need to know the FORTRAN IV programming language. You do not need to know details of internal data formats, machine operation, or the FORTRAN compilation process.

2.2 USING FDT

If you have a program that does not work, follow these steps to begin an FDT debugging session:

1. Compile your FORTRAN program. You must use the FORTRAN compiler option that produces threaded code; FDT does not work with inline code. Do not use the compiler option that suppresses internal statement numbers because FDT needs them. Obtain a source program listing and a storage map listing. You do not need the generated code listing.

2. Link your program units. Include FDT among the input files to be linked. If you are using overlays, place FDT in the root segment of your program. Generate a linker map if your program has named common blocks or assembly language subroutines that you might need to examine.

3. Run your FORTRAN program. FDT takes control and executes an automatic FDT pause before the first executable statement of the program. The following message appears on the terminal:

        FDT V02-01
        FDT PAUSE AT ISN xx IN mainprog
        !

    The variable components of the message are:

        xx   The internal statement number of the first
              executable statement in the FORTRAN program.

mainprog The main program being debugged. FDT refers to
          the main program with the name assigned to it in
          the FORTRAN PROGRAM statement. If you do not use
          the PROGRAM statement to name the program, FDT
          uses the default main program name, .MAIN.

!   The prompt FDT issues to indicate it is ready to
          accept a command.

                              NOTE

          If you do not follow the instructions,
          FDT may not be able to get control, and
          the FORTRAN program will then run
          without FDT. Sometimes FDT gets only
          partial control (as happens when the
          main program is compiled without
          internal statement numbers) and cannot
          continue. In this case, FDT issues the
          message FDT START FAIL, and exits to the
          monitor.

4. Begin debugging your program. Type in the FDT commands that
    you have decided to use to start solving your problem. At
    this point, you must type in at least one command that causes
    an FDT pause, or else the program runs to completion without
    allowing you to enter any FDT commands. If you want to
    execute your program without FDT intervention, type CONTINUE
    or START.

2.3 FDT COMMAND TYPES

There are three classes of FDT commands:

- Program control commands

- Information transfer commands

- FDT control commands

Program Control Commands

Program control commands allow you to control the execution of your
FORTRAN program. You can use the program control commands to halt
program execution, to continue with the next executable statement or
restart from the beginning, to step through the program one or more
statements at a time, or to end a debugging session and exit to the
operating system monitor. The program control commands are START,
STOP, CONTINUE, STEP, PAUSE, RESET, and WATCH.

Information Transfer Commands

Information transfer commands allow you to examine the contents of any
variable or array element in your program and to modify its value. As
part of the command, you can define the data type of the variable, so

that the contents of the variable appear in familiar notation (see Section 2.4.3 on mode codes and Section 2.4.2 on location specification). The information transfer commands are NAME, DIMENSION, TYPE, ACCEPT, and ERASE.

FDT Control Commands

FDT control commands allow you to control the operation of sequences of FDT commands. Using these commands, you can define and execute macros composed of FDT commands, and can branch, conditionally or unconditionally, to another FDT command. The FDT control commands are GOTO, IF, MACRO, and WHAT.

2.4 FDT CONVENTIONS AND TERMINOLOGY

Syntax Conventions

The general form of an FDT command is the command name followed by its parameters. A simple example is

    STEP 3

where STEP is the command name and 3 is the parameter. You can enter a command whenever FDT has issued its exclamation-point prompt.

The syntax conventions for entering FDT commands appear in the following paragraphs:

- Spaces between the prompt and the command are optional. You can place the command directly after the prompt, or you can separate the command from the prompt by any number of spaces.

  The following forms are correct:

      !command
      !   command

- FDT recognizes only the first three letters of any command. You can abbreviate any FDT command name to its first three letters. There cannot be any blanks between the letters in the command name.

  The following examples are correct:

      !START
      !STA
      !STAT   FDT interprets this command as STA.

  The following examples are incorrect:

      !ST A   There is an embedded space in the command.
      !CO   The abbreviation is too short.

- Spaces are required between a command name and its parameter(s). Some FDT commands can accept parameters. You must leave at least one space between the command name and the parameter.

The following examples are correct:

```
!MACRO 1
!PAUSE .MAIN.,20 MACRO 2 AFTER 10
```

The following example is incorrect:

```
!MACRO1
```

- There are two formats for a series of FDT commands. You can
  enter the commands as a list with one command on each line, or
  you can enter several commands on a line, separating the
  commands with semicolons.

  The following example shows two commands on a line:

  ```
  !MACRO 1;  TYPE I,J
  ```

- The maximum command length is one line. There is no
  continuation character to permit a command longer than one
  line. MACRO definitions (but not the MACRO command) can
  continue onto more than one line (see Section 2.5.7).

If you type a command that is incorrectly spelled or incorrectly
spaced, FDT prints the error message ?UNDEFINED and prompts for a new
command.

2.4.1 Current Procedure

An important concept in using FDT is the "current procedure." As its
name implies, the current procedure is the procedure (main program,
subroutine, or function) being executed at a given time. FDT defines
the current procedure as the procedure being executed when an FDT
pause occurs (see Section 2.4.4). You need to know what the current
procedure is when you are defining locations for FDT (see Sections
2.4.2, 2.4.3, and 2.7.2).

2.4.2 The Location Specification

The FDT information transfer commands require you to specify the
location of a variable or array element in the FORTRAN program. The
location specification consists of two parts, the location itself, and
the data type for the location.

There are several ways to specify a location. For a variable, you can
use an offset location, a named location, or a relative location. For
an array element, you use a subscripted name location.

Offset Location

The most direct way of specifying a variable's location is to give its
offset. The offset is the difference between the address of the
variable and the base location of the current procedure's data block.
The offset for any variable appears in the storage map produced by the
FORTRAN compiler.

You specify the offset of a location by typing an octal number:

    nnn

where nnn is the offset of the variable as shown on the FORTRAN storage map. (The offsets in the storage map are expressed in octal bytes.) You need not enter leading zeros.

The following is a fragment from a FORTRAN IV storage map:

FORTRAN IV   Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000322 (105. words)

Name	Type	Offset	Name	Type	Offset	Name	Type	Offset
DAY	I*2	000306	I	I*2	000314	I4	I*2	000312
J	I*2	000316	MONTH	I*2	000304	YEAR	I*2	000310

The offset of the variable DAY is 306; the offset of the variable YEAR is 310.

You can specify the offset of a location only for unsubscripted variables in the current procedure. You cannot use a location offset specification for subscripted array elements, variables in common, or variables in a FORTRAN routine that is not the current procedure. The two exceptions to this rule are for variables in blank common and variables in the main program. (Section 2.7.1 describes special techniques for specifying the offsets of variables in named common.)

1. Variable Offsets in Blank Common

    To specify an offset location for a variable in blank common, type:

        .BCOM.+nnn

    where nnn is the offset of the variable (as shown in the FORTRAN storage map).

2. Variable Offsets in the Main Program

    Any variable in the main program can be referenced at any time by typing:

        mainprog+nnn

    where mainprog is the name of the main program and nnn is the offset of the variable.

Named Location

You can specify the location of a variable by using a name that you have associated with the location. You use the NAME command in FDT to define a name and associate it with a location (see Section 2.5.8). Once you have defined a name, you can use it in any location specification. The name is valid in any procedure.

Relative Locations

You can specify the location of a variable relative to a previously named variable. To specify a location relative to a previously defined name, you enter the name and the displacement of the variable from the named location. This practice is useful primarily for referencing sequences of variables in consecutive locations.

The format of the specification is:

    name+nnn

The components of the specification are:

   name   The named location previously defined in a NAME command.

   nnn   The displacement in octal between the location to be
          specified and the location associated with the name. (The
          displacement is the difference between the address of the
          variable and the address of the named location.)

Subscripted Name Locations

You can specify the location of an array element by using a
subscripted name. The FDT DIMENSION command allows you to define a
subscripted name and associate it with the location of a FORTRAN
array. Once you have defined a subscripted name and associated it
with an array, you can specify the location of any element within the
array with the subscripted name and the appropriate subscripts in
parentheses. For example:

    ARRAY(3,7)

There are two ways to specify the subscript of an array element:

- Use an integer constant.

- Use a name defined in a previous NAME command. If you use a
  name as a subscript, FDT assumes that the variable associated
  with the name has an integer value.

There are certain conventions you must follow for using a subscripted
name location:

- Each subscript value must be within the range defined for that
  dimension. If the subscript value is outside this range, FDT
  prints the following warning message:

      %SUBSCR OUT OF BOUNDS

  Since the message is only a warning, FDT references the
  location using the subscripts specified.

- The number of subscripts you specify in a subscripted name
  must be less than or equal to the number of dimensions in the
  DIMENSION command that defined the name. If the number of
  subscripts is equal to the number of dimensions, FDT uses the
  FORTRAN subscript reference algorithm to locate the array
  element. If the number of subscripts is less than the number
  of dimensions, FDT assumes that the missing subscripts have a
  value of 1. For example, if ARRAY is defined as a
  three-dimensional array, ARRAY(3) is equivalent to
  ARRAY(3,1,1).

- A subscripted name retains its association with a location
  regardless of whether that location is in the current
  procedure. The name loses its association with the location
  only when you redefine the name with another NAME or DIMENSION
  command or when you cancel the name with an ERASE command.

2.4.3 Mode Codes

The second major component of a location specification is the data type or mode for the location.

To specify a mode for a location, type:

    loc[/mode]

The components of the location specification are:

   loc   The location (see Section 2.4.2).

   mode   The FDT mode code for the location. The FDT modes are
           similar to FORTRAN data types. The default mode for a
           location is I, corresponding to the INTEGER*2 data type in
           FORTRAN. Table 2-1 lists the FDT modes and the nearest
           corresponding FORTRAN data type.

Table 2-1
FDT Mode Codes

Mode	FORTRAN Type	Description
I	INTEGER*2	16-bit value displayed in decimal
J	INTEGER*4	32 bits, first 16 bits displayed in decimal
L	LOGICAL*4	32 bits, displayed as T or F
M	LOGICAL*1	8 bits, displayed as T or F
E	REAL*4	32 bits, scientific notation
D	REAL*8	64 bits, scientific notation
C	COMPLEX	64 bits, real and imaginary parts
B	BYTE	8 bits, displayed in decimal
R	----	16 bits, displayed as three RAD50 characters
O	----	16 bits, displayed in octal
An	----	A string of n ASCII characters (where n is in the range 1 to 255)
Z	----	ASCIZ string (as used in the FORTRAN string handling package)
P	----	Dummy variable mode

When you need to indicate that the associated variable is a parameter (dummy variable), you can use the letter P preceding any of the FDT modes. In fact, you must specify this form for any variable listed as a parameter in the attributes section of the FORTRAN storage map.

Examples:

Specification	Meaning
204/E	A REAL*4 variable at offset 204.
16/PI	An INTEGER*2 parameter variable at offset 16.
POWER/C	A COMPLEX variable at a location associated with the name POWER.

Any location specification can include a mode specification. For example:

    TYPE 202/E

The offset of the location is 202 and the data type of location offset 202 is REAL*4.

If a location specification occurs without a mode, there are two possible actions.

   1. If the specification does not involve a name, then FDT assumes the default mode I. For example,

       TYPE 202

   is the same as

       TYPE 202/I

   2. If the specification includes a name (a named location or a subscripted name) then FDT assumes the mode associated with that name. (The mode becomes associated with the name when you define the name in an FDT NAME or DIMENSION command.)

   For example,

       NAME PI,202/E
       TYPE PI

   has the same result as

       TYPE 202/E

   Similarly,

       NAME ANGLE,202
       TYPE ANGLE

   has the same result as

       TYPE 202

The original mode definition can be overridden in subsequent commands. For example, if the name definition is

    NAME PI,202/E

the following commands have the effects shown:

TYPE PI/D   Uses mode D for output.
TYPE PI   Uses originally defined mode E for output.
TYPE 202   Uses the default mode I for output.

The mode code is important for subscripted names in a DIMENSION command because the mode code determines how FDT locates the required array element. You can use the default mode in a DIMENSION command. However, it is better practice to specify the intended mode explicitly in the DIMENSION command.

You can specify a mode only in a location specification. Subscripts and other command parameters are not location specifications, so you cannot associate modes with them.

2.4.4 FDT Pause Definition

FDT contains a pause feature similar in operation to a FORTRAN PAUSE statement: the FDT pause halts execution of the program and allows the program to be continued by further commands. When the pause occurs, you can enter FDT commands. (Do not confuse the FDT pause with the FORTRAN pause. The FORTRAN PAUSE statement cannot cause an FDT pause.)

There are five ways to cause an FDT pause. The pause name indicates the situation causing the pause:

- automatic pause

- entry pause

- statement pause

- step pause

- watch pause

The automatic pause occurs before the first executable statement in the main program. (There is no FDT command for invoking the automatic pause.) The automatic pause occurs only during the first run of the program in a debugging session and not during subsequent runs. That is, subsequent FDT START commands or monitor START or REENTER commands do not invoke the automatic pause.

An entry pause occurs at the entry point of a subroutine or function. You use the PAUSE command to specify the procedure name, and FDT pauses before the procedure begins executing.

A statement pause occurs before execution of a particular FORTRAN statement. Using a PAUSE command, you specify the statement where you want the pause to occur, and FDT pauses when that statement is the next one to be executed.

A step pause occurs after a defined number of FORTRAN statements have executed. You specify the number of statements using the STEP command.

A watch pause occurs when the value of a variable being watched changes. You specify the variable to watch using the WATCH command.

When an FDT pause occurs, FDT prints the following message on the terminal:

    FDT PAUSE AT ISN nnn IN proc
    !

The variable components of the message are:

    nnn   The internal statement number of the next executable
            FORTRAN statement.

    proc   The name of the procedure in which the FDT pause
            occurred. A procedure is a main program, subprogram,
            or function. FDT defines procedure proc as the
            current procedure. This procedure remains the current
            procedure until the next FDT pause occurs.

2.5 DESCRIPTION OF THE FDT COMMANDS

The FDT commands are described in the following section. The commands
are arranged alphabetically for convenient reference. Appendix D
contains a reference summary of the commands.

Useful commands for a new FDT user are:

- PAUSE, STEP, and CONTINUE to control execution

- NAME to refer to variables

- TYPE and ACCEPT to display and change values

ACCEPT

2.5.1 ACCEPT

The ACCEPT command assigns new values to FORTRAN variables. There are three forms of arguments for the ACCEPT command. You can mix all three forms freely within a macro definition. (The third form is not valid outside a macro.) You can specify as many arguments as fit on a single command line. The arguments must be separated by commas.

The first form is:

    ACCEPT loc=value

The argument components are:

    loc   The location whose value is to be changed.

    value   The new value to be assigned to loc. The new value may be a constant in the same data format as the mode of loc, or it may be a previously defined name or subscripted name.

The second form is:

    ACCEPT 'text'

The argument is:

    'text' The literal string to be printed.

This form of the ACCEPT command is identical in function to the analogous form of the TYPE command. The FORTRAN conventions for text literals apply.

The text literal form of ACCEPT is useful for FDT macros, where you can use the text as a prompt to enter new values.

The third form is:

    ACCEPT loc

The argument is:

    loc   The location specification for a value entered from the terminal.

This form of the ACCEPT command is valid only in FDT macros. It requires an input value from your terminal, and prompts for the value with a question mark. The value you enter must follow the same conventions required for other forms of the ACCEPT command.

The following mode conventions apply to the values in the ACCEPT command:

- The value may be a previously defined name. If it is, FDT copies its contents into loc. When you use a name as the value, its mode should match the mode of loc, but need not. If the modes of loc and name are different, the mode of loc determines the number of bytes copied from name into loc (see Table 2-1).

For example:

Mode of loc	Data Type	Number of Bytes Copied
I	INTEGER*2	2
J	INTEGER*4	4
E	REAL*4	4
D	REAL*8	8
Z	ASCIZ	1

When you use a name as a value, FDT ignores the mode of name and uses name only to find the address of the value to be copied. FDT never does conversions from one mode to another.

● String-mode constants (/Z or /An) must appear enclosed in single quotes. The FORTRAN conventions for text literals apply.

● RAD50 constants and logical constants (/R, /L, and /M) must be preceded by the character # to indicate that they do not represent names. For example, the logical constants T and F are represented by #T and #F.

● Complex constants appear as two real constants separated by a comma, in the order real, imaginary.

● Nonstring constants may include at most 40 characters. The number of characters represented by a string constant is limited by the length of the input line.

Examples:

Command	Meaning
ACCEPT 202=1	Sets the INTEGER*2 value at offset 202 to the value 1.
ACC DELTA=EPSILON	Replaces the contents of DELTA with the contents of EPSILON.
ACC 'DELTA=',DELTA	Prints DELTA=? on the terminal and waits for an input value. (Valid only within an FDT macro.)
ACC I=0,J=1,244/E=3.14159	Sets the values of locations named I and J to 0 and 1 respectively; sets the REAL*4 variable at offset 244 to 3.14159.
ACCEPT I=12,'J=',J	Sets the location named I to 12; prints J=? on the terminal and waits for input to set the location named J. (Valid only within an FDT macro.)

NOTE

Both ACCEPT and TYPE use subroutines
that are loaded by FORTRAN to process
FORMAT statements. If you are debugging
a program without D, E, F, or G FORTRAN
format specifications, then some of the
format conversion routines are not
loaded and FDT cannot accept or type
variables with modes C, D, or E. If you
attempt to use modes C, D, or E when the
format conversion routines are not
present, FDT prints the error message
?NO CONVERSION and continues execution.
Section 2.7.3 describes a way to avoid
this problem.

CONTINUE

2.5.2 CONTINUE

The CONTINUE command resumes program execution after any FDT pause.
The next statement executed is the statement whose internal statement
number and procedure name appeared in the last pause message. When it
executes a CONTINUE command, FDT ignores any commands remaining on the
line.

If the current FDT pause occurred as the result of a PAUSE command,
the CONTINUE command can have one optional parameter called the
execution count. FDT ignores the execution-count parameter if the
current pause is an automatic pause or if it occurred as a result of a
STEP or WATCH command.

The form of the command and parameter is:

        CONTINUE [ntimes]

The parameter is:

    ntimes   The execution count. The execution count is an integer
              specifying the number of times the FORTRAN program must
              reach this point before another FDT pause can occur
              here.   If you omit the ntimes parameter or if ntimes =
              1, an FDT pause occurs the next time control reaches
              this point (unless the PAUSE command is replaced or
              cancelled by subsequent commands).

DIMENSION

2.5.3 DIMENSION

The DIMENSION command associates a name and a list of dimensions with the FORTRAN array you want to define, allowing the name to be used as a subscripted name. FDT cannot access FORTRAN virtual arrays. The DIMENSION command is the subscripted-name equivalent of the NAME command.

The form of the DIMENSION command is:

        DIMENSION name(i,j,...)[,loc]

The parameters are:

> name   The FDT name to be associated with the array.
>
> (i,j,...)   The list of dimensions associated with the array. There may be at most seven dimensions. Each dimension value in the list must be an integer in the range 1 through 32767.
>
> loc   The location specification for the first element (base) of the array. The offset for the base of the array appears in the FORTRAN storage map. Offsets for FORTRAN virtual arrays are not valid loc parameters. If you omit the loc argument, FDT erases the name specified (see ERASE command, Section 2.5.4).

The mode code is an important part of the location specification for subscripted names. You can default the mode in a DIMENSION command. However, it is better practice to specify the intended mode explicitly in the DIMENSION command. FDT uses the specified mode in all subscript calculations referring to the array.

ASCII mode (/An) is not valid in a DIMENSION command because it has no meaning when used in subscript calculation. No error message appears if you specify ASCII mode in a DIMENSION command. However, when you attempt to use the subscripted name, FDT responds with the error message ?UNDEFINED.

Examples:

Command	Meaning
DIMENSION DATA(10,10),46/E	A REAL*4 array named DATA with 10 by 10 dimensions at offset 46.
DIM COLUMN(10),DATA(1,3)/E	A one-dimensional REAL*4 array COLUMN equivalenced to the third column of the array DATA.

The following example shows a useful trick for naming one element of
an array.

        NAME W,DATA(5,5)/E   A REAL*4 variable W equivalenced
                                              to array element (5,5) of DATA
                                              (see NAME command in   Section
                                              2.5.8).

2.5.4 ERASE

The ERASE command cancels the association of a name and a location, and frees the space in FDT's internal tables.

The ERASE command has the following form:

    ERASE name1[,name2,...]

You can specify as many names or subscripted names in an ERASE command as can fit on a single line. The names must be separated by commas. Subscripted names must appear without subscripts; subscripts are invalid in the ERASE command. It is not possible to erase the name for part of an array.

Examples:

    ERASE TIME
    ERASE SPEED,DIST,DATA,ICOUNT

ERASE does not change the values of the variables whose associated names are erased.

GOTO

2.5.5 GOTO

The GOTO command changes the order of execution of commands within an FDT macro. It causes an unconditional transfer of control, analogous to that caused by the FORTRAN GO TO statement. (Do not confuse the FDT GOTO and the FORTRAN GO TO; the FDT GOTO cannot change the order of execution of FORTRAN statments.) Like the FORTRAN GO TO, the FDT GOTO requires numeric statement labels.

The form of the GOTO command is:

  GOTO label

(Remember that FDT command names must not contain spaces; GO TO is an invalid FDT command.)

The parameter is:

   label   The numeric label of the command to which control is
            transferred by the GOTO command. The label must be an
            integer in the range 1 to 32767. Embedded spaces are not
            valid.

If two or more labels have the same value, FDT uses the first occurrence of the label. If the label you reference does not exist within the current macro, FDT prints the error message ?LABEL.

You can use the IF command and the GOTO command to create loops of FDT commands analogous to simple FORTRAN loops or to FORTRAN DO loops (see sections 2.5.6 and 2.5.7). For example, if the first command in the loop has the label 100, the last command in a loop might be IF NEXT<>0;GOTO 100. The only loops you can do are on conditions or values; there is no way to increment an index in FDT.

If FDT is executing an infinite loop, you can terminate execution only by typing CTRL/C twice to return to the monitor. The monitor's START or REENTER commands can then operate in the same way as the START command in FDT. However, in some cases, you will have to use the RUN command to reload the FORTRAN program and start the debugging session from the beginning.

Defining a Label

You label a command by preceding it with an integer. The label must follow either the left parenthesis that marks the beginning of the macro definition or the semicolon that delimits the preceding command. The label may be set off by spaces in its defining occurrence.

The following macro definition contains a label:

        MACRO 1(TYPE MAX; 10 ACCEPT 'INIT=',INIT ;IF INIT>MAX;
        GOTO 10; S 3)

2.5.6 IF

The IF command requests conditional execution of another FDT command. You use IF to specify that another FDT command is to be executed only if a given condition is met.

The form of the command is:

    IF loc<rel>value;FDT command

The parameters are:

  loc
    The location specification for the value to be compared. The mode of loc must be E, I, or J.

  <rel>
    The logical relation tested. The parameter <rel> is one of the six logical relations: = (equal), <> (not equal), > (greater than), >= (greater than or equal), < (less than), or <= (less than or equal).

  value
    The value to be compared to loc. The value parameter may be either a constant in the same mode as loc, or a previously defined FDT name. Subscripted names are valid. The mode of the name should (but need not) match the mode of loc. FDT assumes that name has the same mode as loc, and ignores the actual mode of name. It compares the contents of loc and name in the mode defined by loc. FDT never does conversions from one mode to another. (See also the discussion of mixed modes in the ACCEPT command, Section 2.5.1.)

  FDT command
    The single FDT command that is executed only if the logical relation specified between loc and value is true.

The entire IF command must fit on one line.

The IF command compares the value in loc to the value specified. If the relation is true, FDT executes the FDT command specified. If the relation is not true, FDT does not execute the command but skips to the next command.

Example:

    IF PARM/I>5;WATCH COUNT;STEP

If PARM is greater than 5, FDT sets a watch on the location named COUNT, and executes the next FORTRAN statement; otherwise, FDT simply executes the next FORTRAN statement.

MACRO

2.5.7 MACRO

The MACRO command allows you to define, execute, or delete an FDT macro. An FDT macro is a sequence of FDT commands that is executed as a unit. Thus, an FDT macro is analogous to a FORTRAN subroutine or function subprogram.

1. Defining a macro

You can create a new macro, or change an existing macro, with a MACRO command by typing:

    MACRO m(FDT commands)

The parameters are:

> m The number of the macro where m is a value in the range 0 to 7.

FDT commands Any valid FDT command or series of commands.

> There is no space between the macro number (m) and the left parenthesis. The FDT commands appear within the parentheses, separated by semicolons. There is no limit to the number of commands defining a macro. The definition may continue on as many lines as necessary. FDT prompts for each new line. The end of the macro is defined by the closing right parenthesis.

Examples:

The following example prints the values of the variables associated with the names TIME, SPEED, and DIST, and the first two elements of the array named DATA:

    MACRO 3(TYPE TIME,SPEED,DIST,DATA(1),DATA(2))

The following example shows a multiple-line macro definition. The macro accepts a floating-point value VAR from the terminal, and tests whether VAR is within the limits required by the program. If VAR is outside the limits, FDT goes to the command labeled "100" and prompts for another value.

Previous definitions:

    NAME VAR,234/E
    NAME TOP,240/E

Macro definition:

    MACRO 5(100 ACCEPT 'VAR=',VAR; IF VAR<12.; GOTO 100;
    IF VAR>TOP; GOTO 100; CON)

2. **Executing a macro**

You can execute a macro either automatically or manually:

FDT executes a macro automatically whenever it executes an FDT pause that was set up by a PAUSE command containing a MACRO parameter (see Section 2.5.9).

You can execute a macro manually by typing the following command:

    MACRO m

The parameter is:

>   m   The number of the macro to be executed. The value of
>   m is a number in the range 0 to 7.

The specified macro executes until FDT reaches the end of the macro, at which time FDT prompts you for more commands by printing !, or until a START, STEP, or CONTINUE command within the MACRO is executed, at which time FDT resumes execution of the FORTRAN program.

3. **Deleting a macro**

You can delete a macro by redefining it with a null command string. The form of the command is:

    MACRO m()

The parameter is:

>   m   The number of the macro in the range 0 to 7.
>
>   ()   Null command string. The left and right parentheses must
>   not enclose any characters or spaces.

Implicit Macro

MACRO 0 is a special macro number that you cannot associate explicitly with a PAUSE. Any current PAUSE command that you have not explicitly associated with another MACRO is implicitly associated with MACRO 0.

MACRO 0 is usually not defined. If you do define MACRO 0, every PAUSE command without a MACRO parameter executes MACRO 0. You can execute MACRO 0 manually.

NAME

2.5.8 NAME

The NAME command associates a name with a location and a mode. The name must be unique in the first six characters. Any characters after the sixth are ignored. The first character of the name must be a letter. The other characters can be either letters or digits.

The form of the NAME command is:

NAME name[,loc]

The parameters are:

name   The name to be associated with the variable. If the name you specify has already been defined, this NAME command redefines the name. The old definition is lost.

loc   The location specification (as described in Sections 2.4.2 and 2.4.3). If you omit the location specification, FDT cancels the name association (see ERASE command, Section 2.5.4).

Examples:

Command	Meaning
NAME I,202	INTEGER*2 variable I at offset 202.
NAM PARM,16/PI	INTEGER*2 parameter PARM at offset 16.
NAM DELTA,I+2/E	REAL*4 variable DELTA at offset 204. (Relative location I+2 is location 202+2.)
NAM EPSILON,DELTA+4/E	REAL*4 variable EPSILON at offset 210.
NAM PARM	Erases the definition of PARM.

PAUSE

2.5.9 PAUSE

The PAUSE command defines statement pauses and entry pauses (see
Section 2.4.4).

The PAUSE command marks the statement in your program before which you
want an FDT pause to occur. You mark the statement using its internal
statement number. The pause occurs only when the marked statement is
the next one to be executed.

You can define a statement pause for any executable FORTRAN statement
in your program, even if that statement is located in a procedure that
is nonresident or in a shared library. The location of the pause is
stored internally by FDT. It is not stored in the FORTRAN program
itself.

You can place an entry pause on the entry point of any FORTRAN
subroutine or function. Whenever the procedure is called, FDT issues
the message FDT PAUSE AT ISN 0 IN proc. You can use this feature to
detect calls to a procedure without having to determine the first
executable statement of the procedure.

The PAUSE command requires you to specify the location of the pause.
There are two optional parameters. You can specify the optional
parameters MACRO and AFTER in either order. The complete form of the
PAUSE command is:

    PAUSE {[proc],isn} [MACRO m][AFTER ntimes]
           {proc   }

The parameters are:

> proc,isn   The location of the statement to be marked.
>
> proc   The name of the procedure in which the pause
>   occurs.   The default value for the procedure
>   name is the name of the current procedure.   The
>   current   procedure   is defined as the main
>   program, subroutine, or function whose name was
>   printed in the last FDT pause message.
>
>   If you specify a subroutine or function as the
>   procedure and omit the isn parameter, FDT
>   establishes an entry pause at the entry point of
>   the FORTRAN procedure you named. You cannot
>   place an entry pause on the entry point of an
>   assembly language routine. (Note that "proc,0"
>   is not equivalent to "proc" because internal
>   statement number 0 is undefined.)
>
> isn   The internal statement number of the FORTRAN
>   source statement where you want the FDT pause to
>   occur. The internal statement number appears to
>   the left of each statement in the source
>   listing.

NOTE

FDT does not give an error message if
proc,isn specifies a nonexistent or
nonexecutable statement in the FORTRAN
program. The command containing an
invalid internal statement number is
effectively ignored; the nonexistent or
nonexecutable point is not executed by
the program and thus cannot cause an FDT
pause to occur.

MACRO m   The FDT macro that is executed when the FDT
           pause occurs. An FDT macro is a sequence of FDT
           commands that is executed as a unit. It is
           analogous to a subroutine or function in
           FORTRAN. For more information on FDT macros,
           see Section 2.5.7.

           The m argument specifies the number of the FDT
           macro you want to associate with the FDT pause
           you are creating. The macro number must be an
           integer in the range 1 to 7.

           The MACRO m parameter is valid whether or not
           macro m exists. The macro can be defined or
           changed at any time without affecting the PAUSE
           command. If the macro does not exist when the
           pause occurs, FDT operates as if the macro
           parameter were not specified.

           If the macro does exist when the pause occurs,
           FDT executes the macro without issuing an FDT
           pause message. If the macro contains a CONTINUE
           command, execution of the FORTRAN program
           resumes. If the macro does not contain a
           CONTINUE command, FDT prompts for more commands
           when it has finished executing the macro.

           You can abbreviate the word MACRO to MAC. Only
           the first three characters are checked for
           spelling accuracy. (Remember that there must be
           a space between the word MACRO and the macro
           number, m.)

AFTER ntimes   The execution count. The ntimes parameter
                specifies the number of times that the program
                must reach (but not execute) the marked
                statement before an FDT pause occurs at that
                point. The ntimes value must be an integer in
                the range 1 to 32767. The default value for the
                AFTER ntimes parameter is 1, which causes an FDT
                pause the first time program control reaches the
                statement.

FDT decrements the execution count each time control reaches the internal statement number specified. An FDT pause occurs when the count is zero. When the pause occurs, FDT automatically resets the execution count to the default value of 1. If you want to override the default execution count, specify the required execution count in a CONTINUE ntimes command (see Section 2.5.2). (The pause definitions output by the WHAT command contain the current value of the execution count, not the initial value.)

You can abbreviate the word AFTER to AFT. Only the first three characters are checked for spelling accuracy. (Remember that there must be a space between the word AFTER and the execution count, ntimes.)

There can be at most eight active PAUSE commands in a program at any one time. If you attempt to define more than eight pauses in a program, FDT prints out the error message ?NO ROOM.

You cannot place two pause definitions at the same point in your program. If you try to place two pauses at the same point, the second pause definition cancels the first.

Statement pauses are disabled when FDT is in step mode (see STEP command, Section 2.5.12).

Example:

The following command line

    PAUSE SUBR;PAUSE ,10

places an entry pause on procedure SUBR, and a statement pause on statement 10 of the current procedure.

RESET

2.5.10 **RESET**

The RESET command removes pauses created by the PAUSE command.

The form of the RESET command is;

    RESET proc,isn

The parameters are:

      proc,isn   The location in the FORTRAN program for which the pause was defined. The location specified must be exactly the same as that which appeared in the pause definition.

          proc   The name of the procedure in which the pause was to occur.

           isn   The internal statement number of the FORTRAN source statement where the pause was to occur.

For example, the command line

    RESET ,10;RES SUBR

removes the statement pause on statement 10 of the current procedure and the entry pause on procedure SUBR.

START

2.5.11 START

You can issue the START command whenever FDT prints its command prompt. The FDT START command begins executing your FORTRAN program at the first executable statement in the main program. However, START might not work when the system is not in an initial or ready condition. For example, if there are open files, a START command causes an error message and FDT returns control to the monitor.

The START command has no parameters.   When the START command is executed, FDT ignores any commands remaining in the line.

STEP

2.5.12 STEP

The STEP command continues execution of your program in step mode, and indicates the number of statements that you want to execute before the next FDT pause.

The forms of the command and its optional parameter are:

        STEP [n]
        S [n]

(Notice that the STEP command has the special abbreviation S.)

The parameter is:

n   An integer in the range 1 to 32767. FDT executes n FORTRAN statements before pausing. The default value for n is 1, which results in single-step operation with a pause before each statement.

Step mode disables all statement pause definitions. You cancel step mode and reenable the pause definitions by entering any FDT command (other than another STEP command). The occurrence of an FDT pause also cancels step mode.

STEP counts only executable statements. A logical IF statement counts as one executable statement if it contains a GOTO; otherwise, a logical IF statement may be one or two executable   statements (depending on the outcome of the IF test). The END statement is an executable statement. Nonexecutable statements are FORMAT statements, declaration statements, and SUBROUTINE and FUNCTION statements.

After it executes a STEP command, FDT ignores any commands remaining in the line.

STOP

2.5.13 STOP

You issue the STOP command to end a debugging session. The STOP command in FDT is equivalent to a STOP statement in the FORTRAN program. The STOP command closes any open logical units, performs necessary exit actions, and returns control to the operating system.

The STOP command has no parameters.

TYPE

2.5.14 **TYPE**

The TYPE command prints the values of variables in the FORTRAN program during a debugging session.

The forms of the TYPE command are:

> TYPE loc[,more]
>
>   or
>
> TYPE 'text'[,more]

The parameters are:

> loc   The location specification of the variable whose value you want to examine.
>
> 'text' A literal text string to be printed on the terminal. The FORTRAN conventions for text literals apply to FDT literals.
>
> more   Another loc or text parameter.

Multiple loc or text parameters may appear in a single TYPE command. The parameters must be separated by commas. The maximum length of the TYPE command is one line.

FDT finds each location specified and prints the contents of the location in the mode indicated by the location specification. FDT prints text literals exactly as you type them (except for the enclosing quotation marks). If multiple parameters appear in a TYPE command, FDT separates the values with commas.

Examples:

Command	Meaning
TYPE 202	Prints INTEGER*2 variable at offset 202.
TYPE 'HI'	Prints HI.
TYPE 'DON''T'	Prints DON'T.
TYPE DELTA,EPSILON	Prints two values separated by a comma.
TYPE DELTA/O	Prints first 16 bits of DELTA in octal.
TYPE 'DELTA=',DELTA	Prints DELTA= followed by the value of DELTA.
TYPE DATA(I,J)	Prints the value of the (i,j) array element in DATA.

WATCH

2.5.15 WATCH

The WATCH command directs FDT to watch the contents of a location and to perform a watch pause whenever the value in that location changes.

The form of the WATCH command is:

        WATCH [loc]

The parameter is:

> loc   The specification of the location to be watched. (Sections 2.4.2 and 2.4.3 describe how to specify a location.) Only one location can be watched at any given time. A new WATCH loc command cancels the previous WATCH command. A WATCH command without a location specification cancels the previous WATCH loc command.
>
> The location to be watched can have any FDT mode except Alpha (/An). Only the first character of the string is watched for locations in ASCIZ string mode (/Z).

When the value of a watched location changes, FDT prints the message WATCH PAUSE, and performs an FDT pause at the next executable FORTRAN statement. The value must actually change. For example, if IVALUE is equal to 5, then the statement IVALUE=5 does not cause a watch pause. If another FDT command changes the value of a watched location, a watch pause does occur. For example, you can deliberately trigger a watch pause by using FDT's information transfer commands to change the value in a watched location.

Each time a watch pause occurs, FDT cancels the WATCH command that initiated it. You must issue another WATCH command with the same location specification if you want FDT to continue watching the same location.

Watch pauses are independent of other FDT pauses. You can place a watch pause and any other command causing a pause (for example, PAUSE or STEP) on the same FORTRAN statement. If you do specify a watch pause and another pause for the same statement, FDT processes the other pause first. You must resume execution using CONTINUE, STEP, or START before the watch pause can occur.

Examples of the WATCH command:

        WATCH MAX
        WATCH ARRAY(3,2)

WHAT

2.5.16 WHAT

The WHAT command displays the current status of the FDT system.   When
you type WHAT, FDT returns the following information:

- A list of the definitions for all the active pauses. For each
  pause, FDT prints the location of the pause (the procedure
  name and optional internal statement number), the current
  value of the execution count (when it is greater than 1), and
  any associated macro number. The pause listing for an entry
  pause has no internal statement number.

- A list of all currently defined macro numbers and the contents
  of the macros.

2.6 CAUTIONS AND PITFALLS

The following are some general cautions you should observe while using FDT:

- Correct use of spaces is important in FDT.

- Invalid location specifications or constants out of range cause unpredictable results.

- Under the RT-11 SJ operating system, the ends of long lines of text printed at the terminal may be lost when the terminal reaches its right margin. FDT does not provide automatic carriage return or line feed.

- For purposes of subscripting and copying, FDT considers locations with ASCIZ mode (/Z) to have a length of one byte. For example, the command

        ACCEPT NAME/Z=INPUT

    copies only the first character from INPUT to NAME. To copy an ASCIZ string, use mode An, where n is equal to the maximum possible length of the ASCIZ string, including the terminating zero byte. For example, the command

        ACCEPT NAME/A25=INPUT

    copies 25 bytes from INPUT to NAME.

- An FDT error message does not indicate that an entire command was aborted. Some action may have occurred before discovery of the error. In this case, side effects can result. For example, the command

        PAUSE SUBR+12

    produces the error message ?UNDEFINED, but it acts like the command PAUSE SUBR.

- FDT ignores any commands following CONTINUE, START, or STEP. For example, if you type the command line

        STEP 3 ; TYPE INDEX

    FDT executes three FORTRAN statements and then pauses, without typing anything. However, commands following CONTINUE, START, or STEP are executed if FDT control branches around CONTINUE, START, or STEP. For example, if you type

        IF SPY<50;CONTINUE;TYPE INDEX

    then FDT resumes execution if SPY is less than 50, but types the contents of INDEX and waits for another FDT command if SPY is greater than or equal to 50.

- You cannot place an FDT pause on the GO TO of a logical IF statement. For example, suppose the FORTRAN source listing contains the following statements:

        0008   NEXT=NOW+1
        0009   IF(ITEMS(NEXT).GT.ITEMS(NOW)) GO TO 50
        0011   TEMP=ITEMS(NOW)

You cannot place an FDT pause on internal statement 10.   That is, the FDT command

    PAUSE ,10

never causes a pause.

However, an FDT pause on statement 10 would be valid if the source listing contained the following statements:

    0008   DO 50 NEXT=1,N
    0009   IF (ITEMS(NEXT).GT.MAX) MAX=ITEMS(NEXT)
    0011 50   CONTINUE

In this case, the FDT pause occurs before the value of MAX changes if the condition is true. If the condition is false, no FDT pause occurs.

2.7 ADVANCED TECHNIQUES

The following sections describe some advanced techniques that you can use to extend FDT's power.

2.7.1 Named Common

FDT can access variables in a FORTRAN named common block after you define the block using the NAME command. Use the following procedure:

1. Locate the address of the named common block.

    The absolute address of the named common block appears in the link map. The name given to the block in the FORTRAN source code appears in the link map under the column labeled SECTION. The 6-digit number following the name is the address of the block.

    For RT-11 FB foreground programs, the named common block address appearing in the link map is a relative address. The desired absolute address is the sum of the link map relative address and the foreground job load address. Obtain the foreground load address by issuing the FRUN command with the /P option.   (The /P option causes control to return to the monitor; type RESUME to execute the foreground job.)

2. Describe the named common block to FDT.

    A NAME command in the following form defines the block's location:

        NAME block,.ABS.+addr

    The arguments are:

        block   The FDT name of the common block. If the
                block name is the same as the name of a
                variable in a previous NAME command, FDT
                erases the previous name.

        addr   The 6-digit absolute address of the named
                common block.

3. Refer to locations in named common.

   The location of any variable in named common is expressed as follows:

       block+nnn

   The components of the specification are:

       block   The FDT name of the common block.

       nnn   The offset of the variable as given in the FORTRAN storage map.

   You can assign FDT names to variables in named common by using block+nnn as a location in a NAME command.

Examples:

Assume that the name of the FORTRAN named common block is COMBLK and the link map gives its address as 063524.

Describe the named common block to FDT:

    NAME COMBLK,.ABS.+63524

Print the contents of an INTEGER*2 variable at offset 000010 in COMBLK:

    TYPE COMBLK+10

Name the variable at location COMBLK+10:

    NAME INDEX,COMBLK+10

Print the contents of the variable:

    TYPE INDEX

2.7.2 How FDT Generates Addresses

FDT uses the location specification to generate the address and mode information of a FORTRAN variable. The manner in which FDT generates the 16-bit PDP-11 address depends on the information in the location specification. The following section describes the address-generation process for the five kinds of location specifications.

Octal Offsets

The form of the location specification is:

    oct   An octal number in the range of 0 to 177777.

When you give an octal number as a location specification, FDT always interprets the number as an offset from the base address of the current procedure's data block. In other words, FDT adds the octal number to the base address to generate the 16-bit address. Thus, you can use the offsets produced by the FORTRAN compiler to generate addresses for any locations shown in the FORTRAN storage map.

Names

The form of the location specification is:

name   An alphanumeric name unique in the first six characters.

The first character must be a letter. FDT ignores any characters after the sixth. The NAME command associates a name with a location. Once the association has been established, the name refers directly to the location of the variable.

Relative Addressing

The form of the location specification is:

name+oct An alphanumeric name unique in the first six characters, a plus sign, and an octal number in the range 0 to 177777.

This location specification is a form of relative addressing in which the base address is the location associated with the name. FDT adds the octal displacement to the base address to determine the location of the variable. Displacements in the range 100000 to 177777 are negative displacements. FDT assumes that all octal numbers represent two's complement binary integers. For example, the displacement 177776 represents a displacement of -2 bytes (the word preceding the name). You must determine the displacement yourself. It does not appear in the storage map.

You could use relative addressing for array subscripts. However, this is unnecessary because FDT allows you to define subscripted names. Relative addressing allows you to address variables in named common blocks once you have supplied the base address of the common block.

There are several predefined names in FDT that you can use as base addresses for relative addressing. These names are:

.MAIN.   The default name assigned by FDT to the base address of the FORTRAN main program. You can reference any location in the main program using this base. If you named your main program with the FORTRAN PROGRAM statement, then FDT uses that name to find the base address.

.BCOM.   The name assigned by FDT to the base address of the FORTRAN blank common area. Use .BCOM. to reference variables in blank common.

.ABS.   The zero address of memory. FDT uses this name as the base address for referencing any absolute memory location.

For example, the following command prints the contents of absolute location 56 (octal) in octal format:

TYPE .ABS.+56/O

Subscript Addressing

The form of the location specification is:

name(i,j,k,..)   A subscripted name for specifying arrays with at most seven dimensions. The base address for this form of addressing is the address of name(1,1,1,...).

FDT uses the FORTRAN subscripting algorithm to compute the displacement from the base address and to locate the specified array element.

Indirect Addressing

When you define a name as a parameter (by specifying its mode as P), FDT uses the location associated with the parameter name as an indirect address. That is, FDT regards the value associated with the name as the address of the desired value. For example, if the location of the parameter PARM contains 2000, and location 2000 contains 1234, then the command

    TYPE PARM/I,PARM/PI

prints

    2000,1234

2.7.3 Format Conversion Routines

TYPE, ACCEPT, and IF commands use the FORTRAN library format conversion routines. If the program you are debugging does not have D, E, F, or G format specifications, then the linker does not load the routines that FDT requires for C, D, and E modes. If you require C, D, or E modes you can force the linker to load the required routines with the following procedure:

1. When you link your program, include the /I switch in the first linker command line.

2. The linker responds with:

      LIBRARY SEARCH:

   Type RCI$ (followed by a carriage return) to request the floating-point conversion routines and a null line (carriage return only) to end the library search list.

The resulting program contains all the necessary conversion routines.

2.7.4 On-line Debugging Technique (ODT)

If you use assembly language routines with FORTRAN routines, you may sometimes need to use FDT and ODT at the same time. There is no interaction between the two debugging techniques except when ODT, which runs at a higher priority, occasionally interrupts the output of FDT or of the FORTRAN program. FDT does not use breakpoint or T-bit traps.

2.7.5 Execution Speed

FDT uses the FORTRAN internal statement number traceback feature to gain control at the beginning of a FORTRAN program and to execute PAUSE and STEP commands. Therefore, FDT adds some overhead to the execution of each FORTRAN statement and slightly reduces the execution speed of any FORTRAN program.

The amount of FDT overhead time in a main FORTRAN program is difficult to reduce. However, you can get fully debugged, time-critical subroutines to run at full speed. You should compile these routines with the traceback feature disabled. You cannot use FDT within these routines. However, you can use the entry-pause feature of FDT for these routines because the entry pause does not require the traceback feature. If you issue a STEP command immediately before a subroutine compiled without internal statement numbers, control proceeds to the next executable FORTRAN statement in a routine with the traceback feature enabled.

Execution speed with FDT is affected by FDT pauses. Each active pause slows the execution of all FORTRAN statements. It particularly slows those with internal statement numbers greater than that of the statement where the pause occurs. The program runs more quickly if you reset pauses that are no longer necessary. Entry pauses (see Section 2.5.9) do not require overhead for traceback and hence are more time efficient than other PAUSE commands.

APPENDIX A
ARGUMENT AND DEFAULT SUMMARY TABLE

The table contains arguments and default values for the extensions routines. Arguments in square brackets can be either defaulted or omitted. If there is a numeric default value, it appears in the column directly below the argument name. Consult the relevant reference section to find out what happens when you omit a bracketed argument that does not have an explicit default value.

SECTION	ROUTINE					AF
1.4.1	CLRD	iarrayname	npoints	ispace#	rscale	
1.4.2	CVSWG	ivalue	[igain#]			
1.4.3	DIS	iydata	ispace	npoints	istart	increment
1.4.4	DRS	[iarrayname] —	iarraysize	[nsubarrays] 1	[nsamples] iarraysize	[isource] 0	iu
1.4.5	DXY	ixdata	iydata	npoints	istart	increment
1.4.6	FLT16	ivalue				
1.4.7	FSH	iydata	ispace	npoints	istart	increment
1.4.8	FXY	ixdata	iydata	npoints	istart	increment
1.4.9	HIST	iarrayname	iarraysize	nsamples	iendflag#	nleft#
1.4.10	IADC	ichannel	[igain] 1	ivalue#		
1.4.11	IDIR (a)	[isource] 0	iunit	imask	[itype] 0	[iwhere#] —
	(b)	,	iflag	ibefore	iop	[iresult#] —
	(c)	,	iflag	imask	,	
1.4.12	IDOR (a)	[idest] 0	iunit	iselect	iset	[ioutput#] —
	(b)	,	iunit	imask	[iset] —	[iclock#] —
1.4.13	KB2BCD	ibinary				
1.4.14	KBCD2B	idigits				
1.4.15	LED	ivalue	'format'			
1.4.16	LWAIT	ivalue1	ivalue2			
1.4.17	REL	irelay	isetting			
1.4.18	RTS	[iarrayname] —	iarraysize	[nsubarrays] 1	[nsamples] iarraysize	[ifirst] 0	[nch
1.4.19	SDIS					
1.4.20	SETR	[irate] —	imode	rcount	iendflag#	[interval] 1	[com

A-1

APPENDIX B

OBJECT AND SOURCE FILES

LDPOBJ.OBJ, which you receive as part of the FORTRAN extensions software, is a file concatenating 28 individual object files. This appendix lists the names of all the object modules and the names of the corresponding source files. The FORTRAN extensions binary software kit includes one source file, CONFIG.MAC. The other source files can be ordered in a separate kit.

Object File	Source File	Contents
CONFIG.OBJ	CONFIG.MAC	Configuration definition module
CLRD.OBJ	CLRD.MAC	CLRD processor
CVSWG.OBJ	CVSWG.MAC	CVSWG processor
DIS.OBJ	DIS.MAC	DIS, SDIS, and DXY processors
DRS.OBJ	DRS.MAC	DRS processor for all digital I/O units
FSH.OBJ	FSH.MAC	FSH processor
FLT16.OBJ	FLT16.MAC	FLT16 processor
HIST.OBJ	HIST.MAC	HIST processor
IADC.OBJ	IADC.MAC	IADC processor
IDIR.OBJ	IDIR.MAC	IDIR processor
IDOR.OBJ	IDOR.MAC	IDOR processor
KB2BCD.OBJ	KB2BCD.MAC	KB2BCD processor
KBCD2B.OBJ	KBCD2B.MAC	KBCD2B processor
LA0.OBJ	LA0.MAC	Parameter check routine
LA1.OBJ	LA1.MAC	Parameter check routine
LA2.OBJ	LA2.MAC	Parameter check routine
LDISP.OBJ	LDISP.MAC	Completion routine dispatcher
LED.OBJ	LED.MAC	LED processor
LGA0.OBJ	LGA0.MAC	Argument fetch routine
LGA1.OBJ	LGA1.MAC	Argument fetch routine
LICMP.OBJ	LICMP.MAC	Completion of interrupts
LIN16.OBJ	LIN16.MAC	Convert to unsigned binary integer
LINIT.OBJ	LINIT.MAC	Device initialization routine
LPARS.OBJ	LPARS.MAC	Command parser for DIS, DXY, and FSH
LPUT.OBJ	LPUT.MAC	Buffer insertion routine
REL.OBJ	REL.MAC	REL processor
RTS.OBJ	RTS.MAC	RTS processor
SETR.OBJ	SETR.MAC	SETR processor
	LPSMAC.MAC	System macro definitions. LPSMAC must be included in the MACRO assembler command line to assemble any source file except CONFIG.MAC.
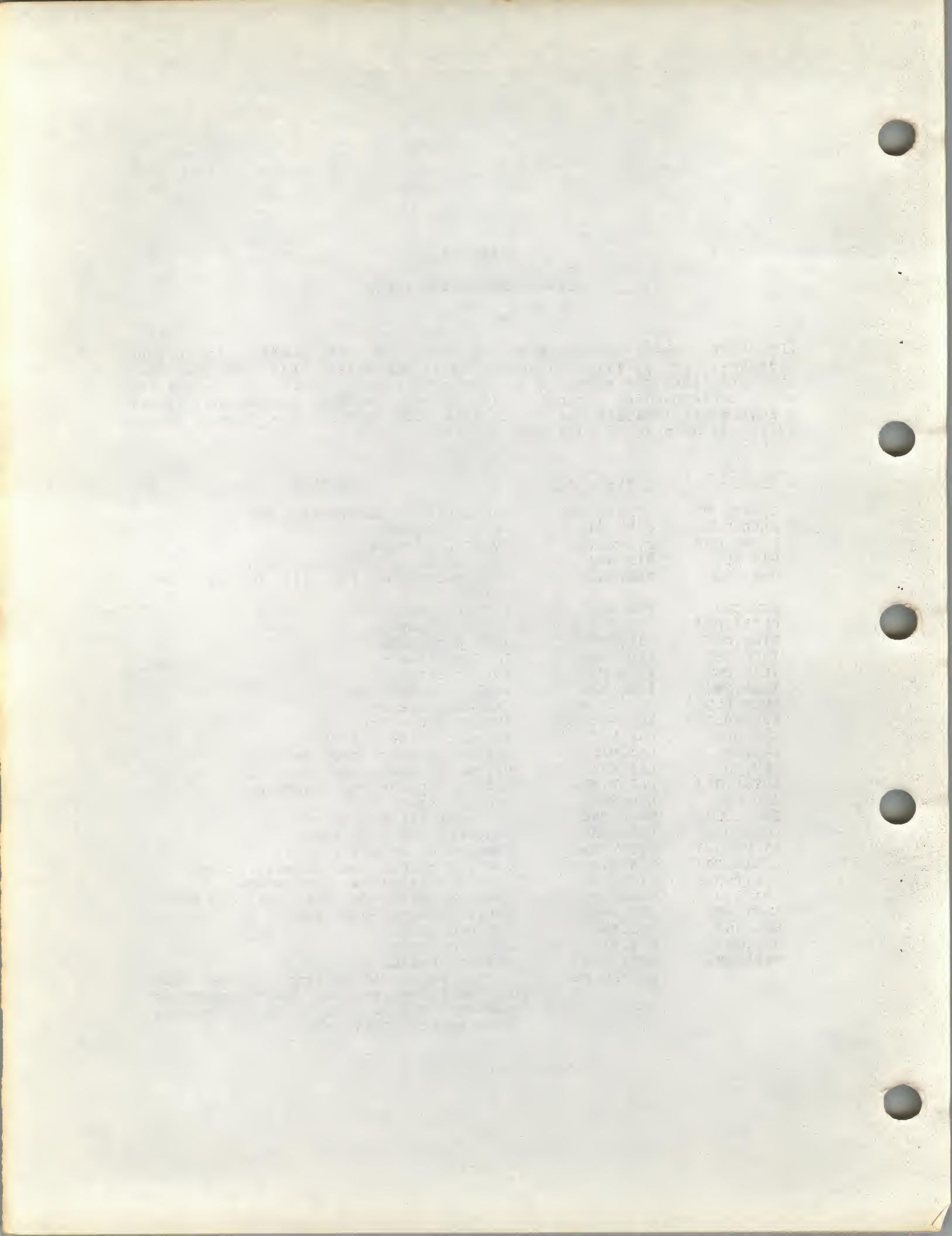
ERROR SUMMARY TABLE

The table summarizes all of the conditions causing errors from the extensions routines. The error name abbreviations (SYN, ARG, DEV, ADC, CLK) are described in Section 1.6.

Routine	Error	Cause
CLRD	SYN	Four arguments required. The number of elements to be scaled (npoints) is less than one. The scale factor (rscale) is negative
CVSWG	SYN	One or two arguments required.
DIS	SYN ARG	Five arguments required. The number of points to display (npoints) is less than one or greater than 4096. The first point (istart) is less than one. The indexing factor (increment) is less than one or greater than 4096.
DRS	SYN ARG DEV	Ten to 12 arguments required. The digital input unit requested (iunit) does not exist. The memory address specified (iunit) is odd. The mode requested (imode) is not valid. The mode requested (imode) is negative, but the call is not in parameter adjustment mode. Digital input unit is in use.
DXY	SYN ARG	Five arguments required. The number of points to display (npoints) is less than 1 or greater than 4096. The first point (istart) is less than 1. The indexing factor (increment) is greater than 4096.
FLT16	SYN	One argument required.
FSH	SYN ARG	Five arguments required. The number of points to display (npoints) is less than 1 or greater than 4096. The indexing factor (increment) is less than 1 or greater than 4096. The first point (istart) is less than 1.

Routine	Error	Cause
FXY	SYN	Five arguments required.
	ARG	The number of points to display (npoints) is less than 1 or greater than 4096.
		The first point (istart) is less than 1.
		The indexing factor (increment) is greater than 4096.
HIST	SYN	Five arguments required.
	ARG	The length of the array (iarraysize) is less than 1.
		The number of samples (nsamples) is less than zero.
IADC	SYN	One to three arguments required.
	ARG	The channel number is greater than 15 (AR11, AD11-K).
		The channel number is greater than 63 (AD11-K with AM11-K, LPS11).
		The gain code (igain) is negative or greater than 4.
	DEV	Autogain was requested for the AR11.
	ADC	A/D converter is in use.
IDIR	SYN	Four or five arguments required.
	ARG	Memory address (iunit) is odd.
		Digital input register (iunit) does not exist.
IDOR	SYN	Four or five arguments required.
	ARG	Memory address (iunit) is odd.
		Digital output register (iunit) does not exist.
KB2BCD	SYN	One argument required.
	ARG	The value (ibinary) is negative or greater than 9999.
KBCD2B	SYN	One argument required.
LED	SYN	Two arguments required.
	ARG	Illegal format requested.
	DEV	System does not have an LPS11.
LWAIT	none	
REL	SYN	Two arguments required.
	ARG	The relay number is less than 1.
	DEV	System does not have an LPS11.
RTS	SYN	10 to 12 arguments required.
	ARG	Starting channel number is too large.
		The number of channels (nchannels) is negative or too large.
		Autogain was requested for the AR11.
		The gain code (igain) is greater than 4.
		The mode (imode) is less than -4 or greater than 15.
		The sum of ifirst and nchannels requests a nonexistent channel.
		Array partition size is not a multiple of nchannels.

Routine	Error	Cause
		Autogain was requested with DMA.
		Continuous sampling with stop code was requested with DMA.
		The array length (iarraysize) with DMA is greater than 4096.
	DEV	A/D converter is in use.
SDIS	none	
SETR	SYN	Four to six arguments required.
	ARG	Rate is less than -4 or greater than 7.
		The mode requested (imode) is not valid.
		The count value (rcount) for AR11 is greater than 255.
	DEV	Clock is running.
	CLK	Clock interrupt lost (clock overrun).
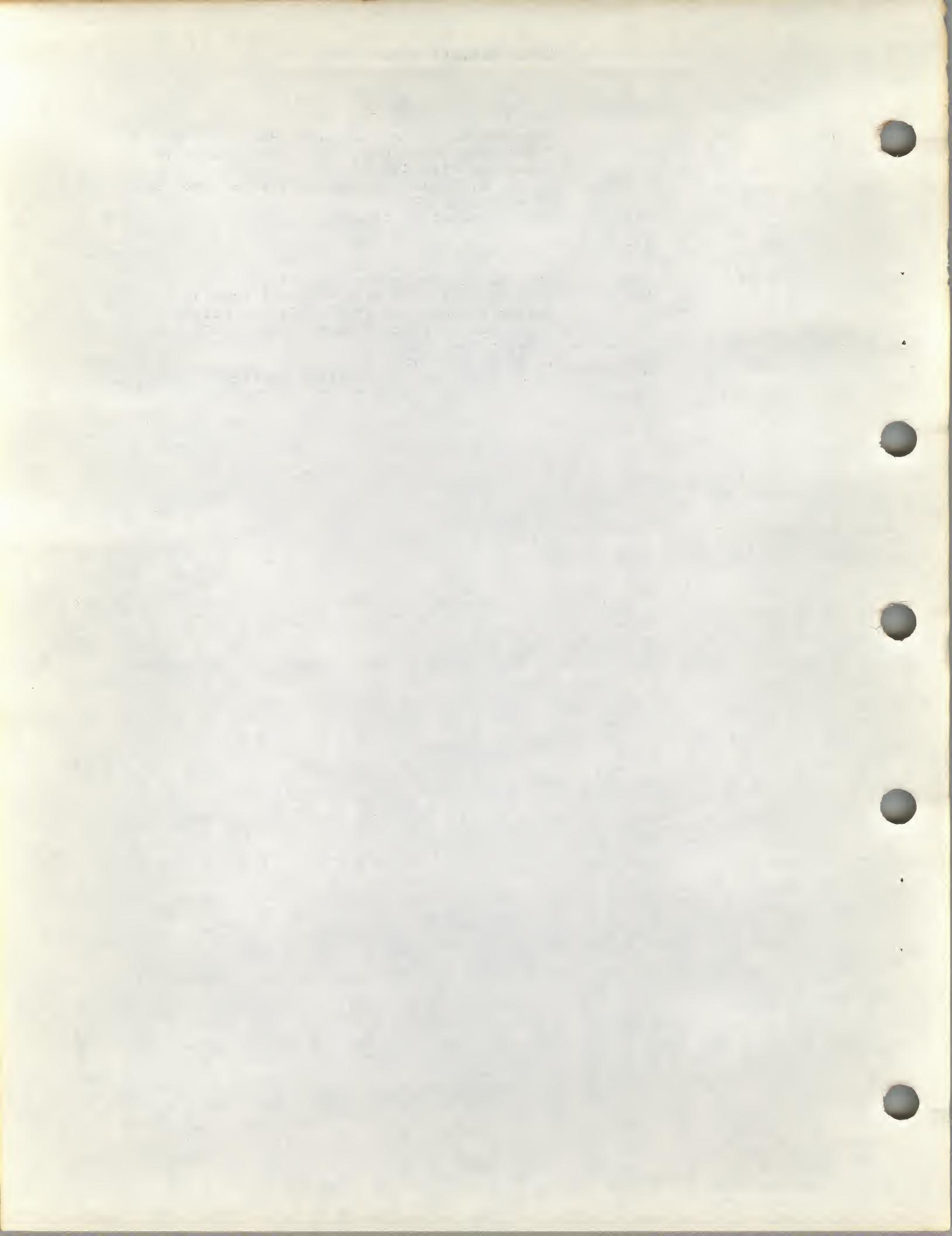
APPENDIX D

FDT COMMAND SUMMARY

Command	Parameters	Description
ACCEPT	loc=value	Assign value as the new contents of loc.
	'text'	Print text on terminal.
	loc	Accept a value from the terminal and assign it to loc.
CONTINUE	[ntimes]	Resume FORTRAN execution.
DIMENSION	name(i,j,...)[,loc]	Associate a location specification and a subscript list with a name.
ERASE	name[,name2,...]	Remove name associations.
GOTO	label	Unconditional branch command changes execution sequence within an FDT macro.
IF	loc<rel>value;FDT command	Execute the FDT command only if the condition is true.
MACRO	m(FDT commands)	Define FDT macro m.
	m	Execute FDT macro m.
	m()	Delete FDT macro m.
NAME	name[,loc]	Associate a location specification with a name.
PAUSE	proc,isn [AFTER ntimes] [MACRO m]	Create an FDT pause at internal statement number isn of procedure proc.
RESET	proc,isn	Remove an FDT pause.
START		Begin execution of main program.
STEP	[n]	Resume execution and execute n statements.

FDT COMMAND SUMMARY

Command	Parameters	Description
STOP		Return to operating system.
TYPE	loc 'text'	Print value or text on the terminal.
WATCH	loc	Cause an FDT pause when the contents of the specified location change.
WHAT		Print FDT status.

APPENDIX E

FDT LOCATION SPECIFICATION FORMATS

Format	Meaning
xxx	Location offset in octal bytes
name	Named location
name+xxx	Relative addressing
name(i,j,k,...)	Subscripted name
.MAIN.	Base address of the FORTRAN main program (if the main program was not named in a PROGRAM statement)
.BCOM.	Base address of FORTRAN blank common
.ABS.	The zero address of memory
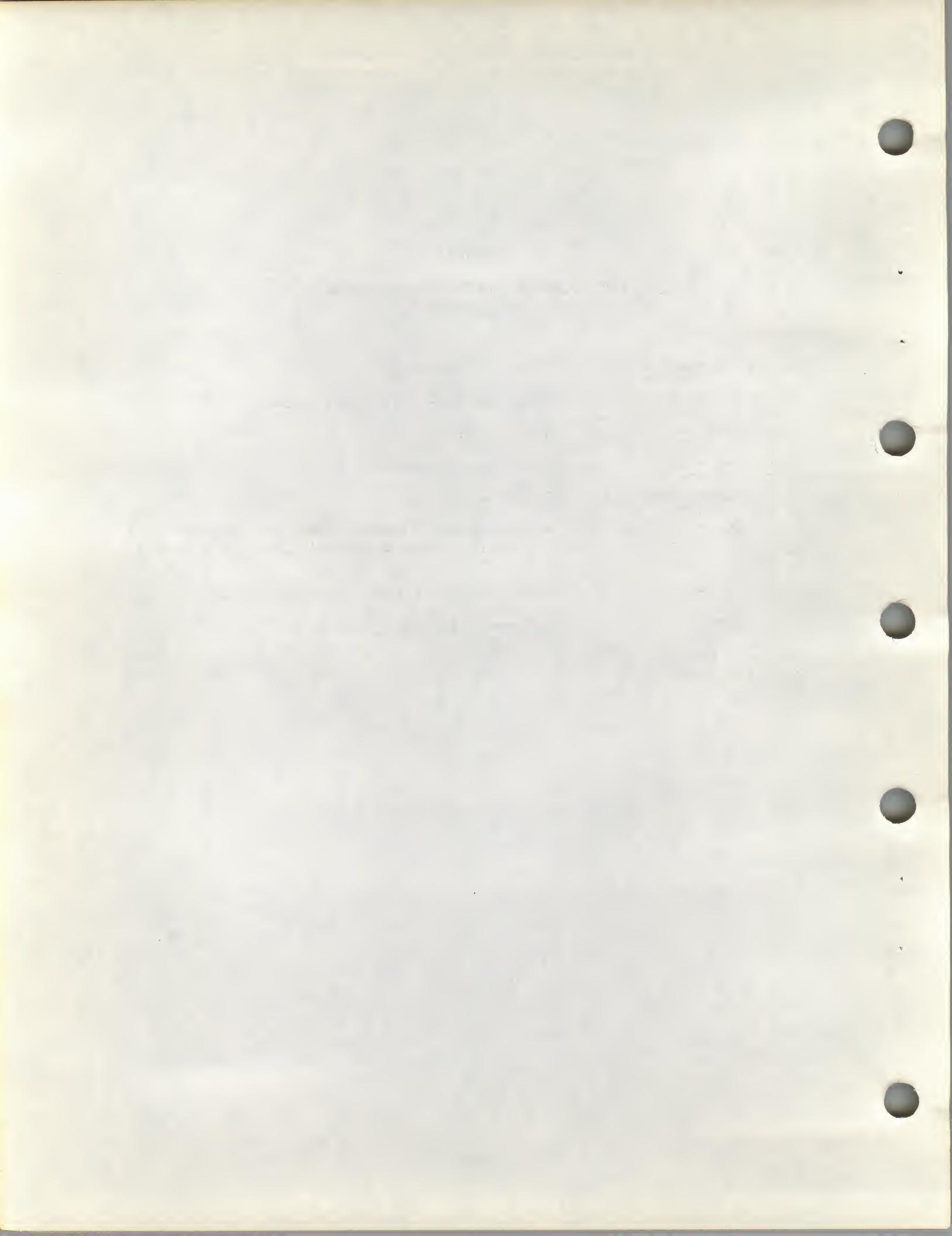
APPENDIX F

FDT MODES

Mode	FORTRAN Type	Description
I	INTEGER*2	16-bit value displayed in decimal
J	INTEGER*4	32 bits, first 16 displayed in decimal
L	LOGICAL*4	32 bits, displayed as T or F
M	LOGICAL*1	8 bits, displayed as T or F
E	REAL*4	32 bits, scientific notation
D	REAL*8	64 bits, scientific notation
C	COMPLEX	64 bits, real and imaginary parts
B	BYTE	8 bits, displayed in decimal
R	----	16 bits, displayed as 3 RAD50 characters
O	----	16 bits, displayed in octal
An	----	A string of n ASCII characters (1 <= n <= 255)
Z	----	ASCIZ string (as used in the FORTRAN string handling package).

Any mode in the above table can be preceded by the letter "P" to indicate that the associated location represents a FORTRAN parameter variable.
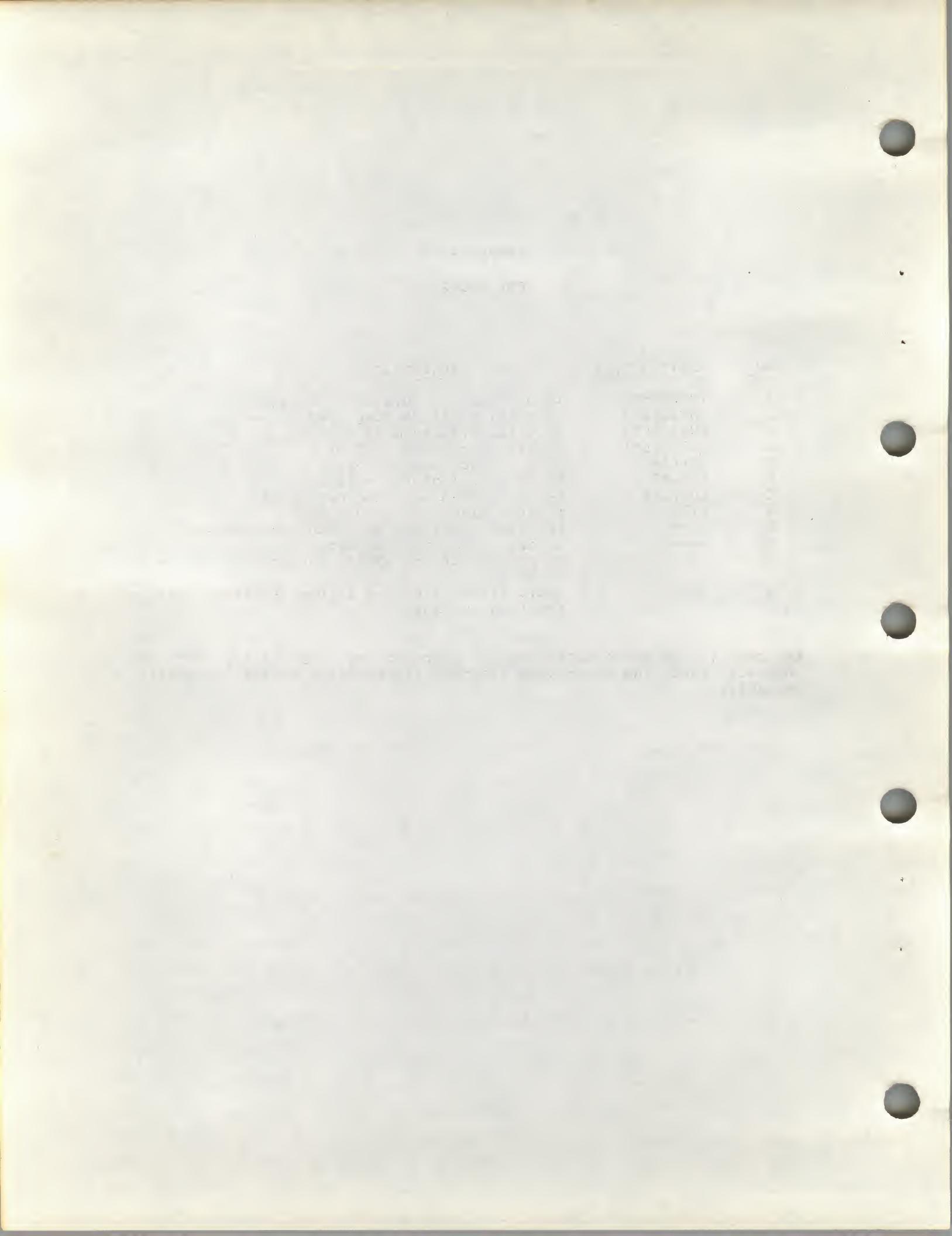
APPENDIX G

FDT ERROR MESSAGES

Message	Description of Error
?BAD DIM	An invalid dimension limit was specified.
?BAD LOC	The location specification is not aligned correctly according to its mode or it references a location outside the program bounds.
?BAD MACRO	An attempt was made to redefine a macro while it was executing.
?BAD SUBS	Subscripts are in an invalid format or cause overflow.
FDT START FAIL	Invalid main program, bad start address, or internal statement numbers not enabled in main program.
?FORMAT	The input constant is not in the expected mode.
?LABEL	The label specified does not exist.
?MACRO #	The macro number is not in the range 0-7.
?NO CONVERSION	Floating-point formats are not available with current FORTRAN program (see Section 2.7.3).
?NO ROOM	Not enough memory space to define a new PAUSE, MACRO, or NAME.
?ONLY IN MACRO	The operation attempted is valid only within an FDT macro.
?PAUSE NOT FOUND	An attempt was made to remove a PAUSE that is not active.
%SUBSCR OUT OF BOUNDS	The specified subscripts exceed the declared dimensions. (Warning message only.)
?UNDEFINED	Syntax error or undefined name.

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will
use comments submitted on this form at the company's
discretion. Problems with software should be reported
on a Software Performance Report (SPR) form. If you
require a written reply and are eligible to receive
one under SPR service, submit your comments on an SPR
form.

Did you find errors in this manual? If so, specify by page.

_____
_____
_____
_____
_____
_____

Did you find this manual understandable, usable, and well-organized?
Please make suggestions for improvement.

_____
_____
_____
_____
_____
_____

Is there sufficient documentation on associated system programs
required for use of the software described in this manual? If not,
what material is missing and where should it be placed?

_____
_____
_____
_____
_____
_____

Please indicate the type of user/reader that you most nearly represent.

☐ Assembly language programmer
☐ Higher-level language programmer
☐ Occasional programmer (experienced)
☐ User with little programming experience
☐ Student programmer
☐ Non-programmer interested in computer concepts and capabilities

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
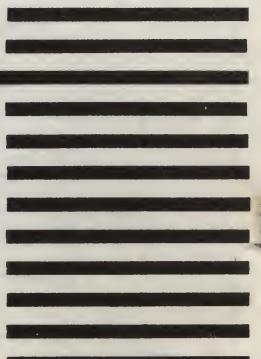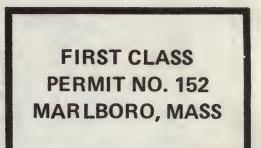   or
                                              Country

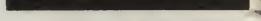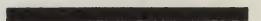Please cut along this line.